ULBS
Universitatea "Lucian Blaga" din Sibiu

Şcoala doctorală interdisciplinară
Domeniul de doctorat: Calculatoare și Tehnologia Informației

TEZĂ DE DOCTORAT

SISTEM INTEGRAT PENTRU MAPAREA CVASI-OPTIMALĂ A UNOR APLICAȚII ȘTIINȚIFICE PE ARHITECTURI HPC PARAMETRIZABILE

— REZUMAT —

Doctorand:
Ing. ION-DAN MIRONESCU

Conducător științific:
Prof. univ. dr. ing. LUCIAN N. VINȚAN
Membru titular al Academiei de Științe Tehnice din România

SIBIU 2018

CUPRINS

TEZA DE DOCTORAT

Cuvinte cheie

HPC calcule de înaltă performanță; MDE Inginerie bazată pe model; LBM metode Lattice Boltzmann; calcul paralel și distribuit; algoritmi de planificare; optimizare multicriterială; modelare și simulare cu rețele Petri.

Lista tabelelor

Lista figurilor

REZUMATUL
TEZEI DE DOCTORAT

1. INTRODUCERE

Ne propunem să dezvoltăm o metodologie unificată care să permită identificarea soluției optime de mapare a unor aplicații științifice numerice complexe, pe o arhitectură hardware de calcul de înaltă performanță, generică, puternic parametrizabilă, de tipul *High Performance Computer* - HPC (sistem de calcul eterogen de înaltă performanță) încă din faza de pre-proiectare a ansamblului hardware–software (co-design). Optimalitatea presupune folosirea cât mai eficientă a resurselor disponibile, în scopul obținerii unei performanțe maxime, în condiții restrictive de energie consumată, complexitate etc., fiind deci o problemă multicriterială complexă [Vin16b].

Obiectivele cercetării prezentate în această lucrare au fost următoarele:

- Dezvoltarea unei platforme care să suporte procesul integrat de generare a variantelor de mapare, de evaluare a performanțelor acestor variante și de selecție a variantei optime. Acest proces va folosi instrumentele *Model Driven Engineering* pentru gestiunea eficientă a variantelor de mapare și a artefactelor (modele executabile, cod) corespunzătoare;
- Definirea unei arhitecturi de aplicație software, care să se poată adapta la particularitățile diferitelor niveluri de organizare ale arhitecturii hardware, astfel încât să mapeze cât mai bine concurența exprimată în software, pe paralelismul disponibil în hardware;
- Modelarea holistică a arhitecturii hardware–software la un nivel de detaliu, care să permită evaluarea gradului în care aplicația exploatează paralelismul hardware-ului atât la nivelul nodului, cât și la nivelul rețelei. Modelul trebuie să surprindă și interacțiunea dintre cele două niveluri ierarhice;
- Verificarea și evaluarea, prin simulări complexe, a arhitecturii hardware–software propuse, pentru validarea modelului și arhitecturii;
- Optimizarea multi-obiectiv a arhitecturii hardware–software modelate și implementate, într-un cadru unitar de analiză.

Pentru evaluarea performanțelor diferitelor variante de mapare în faza de pre-proiectare, avem nevoie de un model al ansamblului hardware–software, care să permită estimarea acestor performanțe, în lipsa codului aplicației și a hardware-ului, aflat în faza de concept (arhitectură HPC generică, parametrizată). Modelul trebuie să reprezinte ansamblul în mod uniform, astfel încât:

- maparea să poată fi descrisă prin asocierea componentelor concurente ale aplicației de tipul *Computational Fluid Dynamics* (CFD) cu unitățile de execuție de la nivelul nodului;
- să se poată evidenția în ce măsură maparea exploatează paralelismul existent în hardware.

Pentru a susține acest demers, vom implementa metodologia unificată într-un mediu integrat, care se va baza pe modelarea uniformă, cu rețele Petri, a ansamblului hardware–software. În scopul exploatării paralelismului în mod dinamic, modelul va avea gradul de detaliere necesar pentru a surprinde atât comportamentul componentelor concurente ale

aplicației software, cât și pe cel al unităților de execuție pe care acestea sunt alocate.

Față de simulatoarele *cycle accurate*, care, de obicei, sunt limitate la nivelul procesorului, metodologia noastră permite o abordare holistică a nivelurilor rețea, nod eterogen de procesare (CPU, GPU) și procesor.

Un alt avantaj al metodei de modelare cu rețele Petri este că permite verificarea formală a modelelor exprimate în acest formalism; acest lucru nu este posibil cu modelele din simularea de tip *cycle driven* (simulare de tip *execution driven* cu informații arhitecturale în fiecare ciclu de tact CPU), de obicei implementate prin cod (în limbaje de nivel mediu / înalt).

Pentru că modelele au o reprezentare grafică, se poate dezvolta un cadru unificat de modelare/ simulare/ verificare/ optimizare, care nu poate fi construit în jurul simulatoarelor detaliate, la nivelul *cycle accurate*. Rețelele Petri nu pot, însă, simula cu aceeași precizie ca și simulatoarele *cycle accurate*; de aceea, metodologia propusă în cadrul tezei este adecvată fazei de pre-proiectare. Estimarea performanțelor prin simulare cu rețele Petri permite identificarea unei clase de soluții implementabile, care trebuie apoi rafinate, la nivel arhitectural, folosind simulări tip *cycle accurate* și metode de *Design Space Exploration* clasice.

În continuare, se prezintă succint conținutul fiecărui capitol al lucrării.

Capitolul 2 prezintă stadiul actual al domeniului, care evidențiază:

- posibilitățile oferite de instrumentele folosite în *Model Driven Engineering*, pentru integrarea procesului de generare a variantelor constructive;
- particularitățile sistemului hardware curent și evoluția acestui sistem, pentru a stabili arhitectura care va fi modelată, precum și elementele care trebuie parametrizate;
- particularitățile suportului software folosit pentru paralelizarea aplicațiilor;
- metodele de optimizare care pot fi folosite pentru obținerea mapării dorite;
- metodele de modelare și simulare care pot fi folosite pentru evaluarea performanțelor.

Capitolul 2 se încheie cu o sinteză (Secțiunea 2.4), care conține deciziile adoptate și oportunitățile identificate pentru contribuții originale la dezvoltarea domeniului.

Capitolele 3, 4, 5 și 6 reprezintă contribuția originală în domeniu a autorului acestei lucrări. Astfel, capitolul 3 prezintă modelele și transformările implementate în cadrul tezei cu ajutorul instrumentelor *Model Driven Engineering*. Modelele generate la acest nivel sunt modele descriptive, exprimate în limbajul de descriere a sistemelor *System Modelling Language* (SysML). Modelele descriptive permit gestiunea eficientă a diferitelor variante investigate și aplicarea modificărilor impuse de procesul de optimizare. Pe baza lor, se generează modelele executabile pentru variantele de mapare.

Capitolul 4 prezintă arhitectura originală propusă pentru aplicația software, care implementează metoda numerică din familia Lattice Boltzmann (Lattice Boltzmann Methods LBM). Arhitectura este bazată pe combinarea mediilor de execuție a două platforme software pentru dezvoltarea de aplicații concurente, CHARM++ și StarPU. Fiecare din cele două platforme mapează mai bine concurența pe unul din cele două niveluri de organizare ale hardware-ului High Performance Computing (HPC). Astfel, modelul de paralelism folosit de limbajul CHARM++ exploatează mai bine rețeaua de noduri, iar mediul StarPU permite exploatarea mai eficientă a fiecărui nod eterogen. Prin combinarea celor două platforme, rezultă o arhitectură de aplicație care permite exploatarea eficientă a ambelor niveluri. În arhitectura propusă în cadrul tezei, aplicația de simulare cu lattice Boltzman este compusă din task-uri OpenCL și obiecte CHARM++ (*chare*-uri). Task-urile OpenCL implementează partea

de calcul a algoritmului numeric de simulare. *Chare*-urile implementează partea de comunicație a aceluiași algoritm printr-un protocolul de negociere.

Capitolul 5 prezintă infrastructura originală dezvoltată în cadrul tezei pentru maparea dinamică a aplicației pe arhitectura hardware. Soluția propusă combină un planificator dinamic distribuit (care echilibrează sarcina computațională între nodurile rețelei) cu un planificator dinamic bazat pe liste (care distribuie sarcina computațională la nivel de nod). Planificatorul de la nivel de rețea se bazează pe un protocol de negociere între *chare*-uri, pentru a implementa un algoritm difuziv de transfer de sarcină (*work stealing*). Planificatorul de la nivel de nod folosește un algoritm de tip *Heterogeneous Earlier Finish Time* (HEFT) (pentru a alege o unitate de procesare pe care să distribuie task-ul) și un algoritm local de transfer de sarcină (pentru a redistribui task–ul, la nevoie).

Capitolul 6 prezintă modelele executabile dezvoltate de autor pentru hardware și pentru toate nivelurile software implicate în arhitectura aplicației: cele două medii de execuție, infrastructura de mapare dinamică, componentele aplicației (*chare*-uri și task-uri).

Folosind rețele Petri colorate temporizate, a fost dezvoltat și testat modelul complet hardware–software al mapării aplicației pe sistemul HPC generic, parametrizat.

Folosind modelul hardware al unei rețele de noduri omogene, au fost simulate mai multe variante de mapare statică a distribuției sarcinii pe noduri. Simulările au fost comparate cu cele ale simulatorului BigSim, folosit pentru estimarea performanțelor aplicațiilor CHARM++ pe sisteme tip HPC. Rezultatele au arătat o bună concordanță calitativă între cele două simulări.

Folosind un nod eterogen, a fost simulat comportamentul algoritmilor de mapare dinamică, implementați în mediul StarPU, cu diferite tipuri de sarcină. Rezultatele au fost comparate cu rezultatul rulării aplicației reale pe arhitectura modelată. Simulările au permis un studiu comparativ al diferiților algoritmi de mapare. De asemenea, au confirmat faptul că modelul propus poate fi folosit pentru evaluarea unui astfel de algoritm încă din faza de pre-proiectare.

Prin intermediul mediului integrat, modelul a fost transformat într-o reprezentare, folosind rețele Petri temporizate extinse. Modelul rezultat a fost folosit pentru testarea prin simulare a algoritmului de echilibrare a încărcării pe două niveluri, prezentat în Capitolul 5. Folosind pentru task-uri timpii de execuție obținuți prin măsurare și prin simulare cu simulatorul Multi2Sim, s-a estimat o reducere a timpului total de execuție:

- cu 15%, dacă se folosesc planificatorul de la nivelul rețelei și cel de la nivelul nodului;
- cu 8,6%, dacă se folosește doar planificatorul de la nivelul nodului

față de cazul în care se folosește doar o distribuție statică la ambele niveluri.

Pe baza datelor din literatură ([Calore2017] [Martinez2009] [van Werkhoven2014]), pentru un cluster cu 16 noduri heterogene care rulează aplicația numerică de calcul, am dezvoltat:

- un model matematic original, care estimează timpul necesar și consumul energetic la calculul unei sarcini computaționale date, pentru fiecare nod;
- un algoritm euristic original, care aproximează setul Pareto al problemei de minimizare a timpului de execuție, energiei consumate și încărcării la calculul unei sarcini computaționale date, distribuite pe tot clusterul.

Modelul a fost folosit pentru testarea prin simulare a algoritmului de mapare cvasi-

optimală descris în Capitolul 5 și adaptat pentru optimizarea multicriterială. A fost testată capacitatea de redistribuire dinamică a algoritmului în situații normale (iterații curente) sau speciale (inițializare, creșterea sarcinii datorită rafinării discretizării, creșterea sarcinii la căderea unui nod). S-a arătat că algoritmul de mapare are capacitatea de a trata toate aceste situații și generează o mapare în apropierea aproximării setului Pareto.

Capitolul 7 prezintă concluziile generale ale tezei: gradul de rezolvare a obiectivelor propuse, contribuțiile originale şi perspectivele de cercetare.

2. FUNDAMENTARE TEORETICĂ

În acest capitol s-au prezentat în mod sintetic, pe baza unei cercetări bibliografice laborioase, următoarele aspecte mai importante:

- Metoda *Model Driven Engineering* se folosește în domeniul sistemelor incorporate (emebedded) pentru dezvoltarea simultană hardware–software (co-design), dar nu la dezvoltarea aplicațiilor pentru *High Performance Computing*, deși se pune și aici problema dezvoltării simultane din punct de vedere hardware–software. Ca urmare, considerăm că demersul nostru în această direcție este oportun;
- Arhitectura hardware cea mai frecventă pentru sistemele din Top 500 [top500] este cea de cluster cu noduri legate prin rețele InfiniBand sau Ethernet în topologie Fat tree. Dintre aplicațiile științifice, cele bazate pe metoda numerică Lattice Boltzmann pot exploata mai bine astfel de arhitecturi. Există relativ puține aplicații dezvoltate în acest domeniu;
- Platformele software care permit dezvoltarea de aplicații pentru sisteme High Performance Computing s-au dezvoltat istoric în jurul unui singur model de programare paralelă (transfer de mesaje sau memorie partajată), și anume cel care se potrivea cel mai bine cu platforma pentru care au fost dezvoltate inițial. Odată cu apariția procesoarelor cu nuclee multiple și a acceleratoarelor, platformele trebuie să se adapteze la structura de cluster cu noduri multiprocesor eterogene, prin integrarea ambelor modele de programare paralelă. Nici o soluție actuală nu o face, însă, complet satisfăcător. Aici identificăm o oportunitate pe care dorim să o exploatăm prin arhitectura pe care o propunem în Capitolul 4;
- Din analiza metodelor de planificare și a caracteristicilor aplicației, am concluzionat că trebuie să folosim o planificare distribuită a task-urilor concurente, pentru a permite scalarea sistemului. De asemenea, pentru a face față unui mediu care se poate schimba, vom folosi o planificare dinamică adaptivă. Acest lucru ne permite să abordăm și problema asigurării fiabilității sistemului de calcul în dezvoltarea algoritmului propriu de planificare (prezentat în Capitolul 5). De obicei, în algoritmii de planificare existenți nu sunt incorporate și proceduri care să permită continuarea calculelor, în cazul în care unul dintre noduri se defectează (asigurarea/ creșterea fiabilității sistemului prin tehnici de toleranță la defectări), ceea ce constituie un element de originalitate introdus de lucrarea de față;
- Cea mai potrivită metodă de simulare este cea cu evenimente discrete, pentru că permite reducerea timpului de simulare. Din această categorie, rețelele Petri oferă expresivitate, ușurință în dezvoltare, precum și posibilități de verificare formală. Nu există un model

comun hardware–software, la același grad de detaliere, pentru dezvoltarea, simularea și evaluarea sistemelor HPC, ceea ce conferă originalitate demersului nostru, pe care îl vom prezenta în Capitolul 6.

3. MEDIUL DEZVOLTAT PENTRU INGINERIA BAZATĂ PE MODEL

În acest capitol au fost prezentate instrumentele dezvoltate în cadrul tezei care permit aplicarea metodelor de tip *Model Driven Engineering (MDE*, pentru maparea optimă a unei aplicații pe un sistem de calcul HPC parametrizabil. Această abordare, folosită la dezvoltarea hardware–software a sistemelor integrate (embedded), este originală în contextul dezvoltării sistemelor HPC.

Instrumentele dezvoltate (modelele descriptive formulate în limbajul SysML, transformările între modele M2M, transformările de la model la text M2T) sunt prezentate în figura 6. Instrumentele permit gestiunea mai eficientă a procesului de generare a variantelor constructive. În faza de pre-proiectare, în care trebuie generate și evaluate multe astfel de variante, acest lucru constituie un avantaj important.

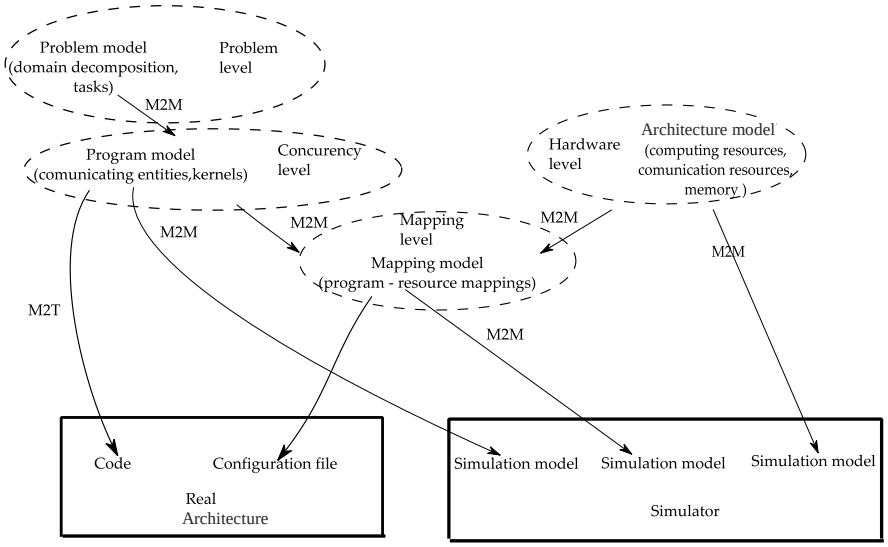


Figura 6. Nivelurile de abstractizare ale modelelor [Mironescu 2011]

Dacă s-ar administra separat sursele și fișierele de configurare pentru aplicație și fișierele cu modelele executabile specifice fiecărui simulator, orice schimbare necesară pentru obținerea unei noi variante ar trebui operată în fiecare din aceste fișiere; acest lucru necesită efort și procesul este predispus la erori. Prin folosirea metodelor MDE, orice schimbare se operează o singură dată în modelul descriptiv și se propagă prin lanțul de transformări, pentru a obține toate fișierele corespunzătoare noii variante.

Pe același principiu, se poate folosi o ierarhie de simulatoare pentru aceeași configurație de mapare. Rezultatele obținute cu unul dintre simulatoare pot fi propagate la toate modelele executabile și pot fi astfel folosite în celelalte simulatoare.

Sistemul poate fi extins ușor să folosească un alt simulator, prin scrierea unei transformări care generează fișierul cu modelul executabil din modelul descriptiv.

4. ARHITECTURA APLICAȚIEI SOFTWARE

Aplicația folosește descompunerea domeniului (pentru paralelizare) și metoda zonelor fantomă (pentru asigurarea consistenței soluțiilor calculate pe subdomenii).

Arhitectura propusă în acest capitol pentru aplicația software combină, în premieră, două medii de execuție (StarPU si CHARM++) pentru programe concurente. Fiecare dintre aceste medii este conceput pentru exploatarea eficientă doar a unuia din cele două niveluri de organizare ale unui sistem HPC și anume:

- la nivelul rețelei de noduri, CHARM++ facilitează distribuirea aplicației între nodurile care comunică prin rețeaua de interconectare;
- la nivelul nodului eterogen, StarPU facilitează distribuirea aplicației pe unitățile de procesare eterogene ale unui nod.

Aplicațiile curente sunt dezvoltate în jurul unei singure platforme (CHARM++ sau StarPU), așa că vor avea, în mod inerent, limitările acesteia.

Față de o aplicație dezvoltată folosind doar CHARM++, aplicația software cu arhitectura propusă în acest capitol va avea avantajul distribuirii mai eficiente pe sisteme hardware eterogene, pe care *chare*-urile (obiectele alocabile din CHARM++) nu se pot mapa ușor. De altfel, dezvoltatorii CHARM++ au identificat această problemă. Soluțiile propuse și în curs de implementare sunt folosirea de task-uri similare cu cele din StarPU [Robson2016], sau integrarea OpenMP [Bak2018]; acestea nu funcționează pe o gamă la fel de mare de dispozitive și nu au un mecanism extensibil de echilibrare a încărcării, precum cele disponibile în StarPU.

Față de o aplicație dezvoltată folosind doar StarPU (sau folosind un concept similar de task, de exemplu modelul de programare OmpSs), arhitectura propusă de autorul acestei teze permite implementarea unor mecanisme de comunicație între noduri mai sofisticate și mai eficiente, bazate pe modelul Actor suportat de CHARM++. Acest avantaj a fost folosit pentru implementarea unui sistem de echilibrare a încărcării între noduri, bazat pe un protocol de negociere, care va fi prezentat în Capitolul 5.

Arhitectura propusă permite separarea codului aplicației pe trei niveluri:

- Rutinele OpenCL, în care este implementat algoritmul numeric;
- Codul StarPU, în care este implementat planificatorul pentru distribuirea task-urilor pe nod;
- Codul CHARM++, în care este implementată comunicarea între noduri.

Arhitectura propusă de noi permite dezvoltarea și optimizarea separată a aplicației software pe cele trei niveluri. Astfel, rutinele OpenCL pot fi editate independent și apoi rulate pe arhitectura reală sau pe simulatoare de tipul *cycle by cycle,* cum este Multi2Sim. Rutinele pot fi îmbunătățite pe baza rezultatelor rulării. Un exemplu este prezentat în capitolul 6.2. Codul sursă rezultat este asociat cu modelul din platforma de *Model Driven Engineering.*

În comparație, aplicațiile dezvoltate folosind biblioteca OpenMP (sau platforme care folosesc OpenMP pentru distribuirea sarcinii pe nodurile eterogene, inclusiv CHARM++) integrează porțiunea de cod care este paralelizată direct în codul aplicației, iar rularea și testarea independentă a porțiunii de cod este mai complicată. Față de aceste aplicații, complexitatea aplicației propuse în acest capitol este mai mare, pentru că există mai multe fișiere care trebuie gestionate, dar componenta de Model Driven Engineering poate gestiona cu succes această complexitate.

La nivelul StarPU, se poate face dezvoltarea și testarea planificatoarelor care distribuie sarcina computațională pe nodul eterogen, fără ca acest proces să afecteze celelalte două niveluri. În acest mod au fost investigate prin simulare performanțele planificatoarelor implementate în StarPU în distribuirea sarcinii computaționale a aplicației, proces prezentat în capitolul 6. Rezultatele simulării au fost folosite pentru dezvoltarea propriului planificator pentru nivelul nodului eterogen, prezentat în capitolul 5.

La nivel CHARM++ se poate dezvolta și testa mecanismul prin care se face schimbul de date între noduri prin definirea metodelor obiectelor CHARM++(*chare*-uri). Implementarea are loc doar la nivelul surselor CHARM++ fără să afecteze celelalte două niveluri. În acest mod, a fost dezvoltat mecanismul care asigură comunicarea necesară algoritmului numeric și apoi protocolul de comunicare care implementează planificatorul dinamic difuziv de la nivelul rețelei de noduri, prezentat în capitolul 5.

5. INFRASTRUCTURA PENTRU OPTIMIZAREA ÎNCĂRCĂRII

Acest capitol prezintă algoritmul de mapare cvasi-optimală dezvoltat în cadrul tezei pentru echilibrarea distribuției sarcinii computaționale a aplicației pe sistemul HPC generic. Algoritmul de mapare cvasi-optimală combină doi algoritmi, după cum urmează:

- un algoritm de distribuție dinamică a sarcinii computaționale între nodurile unui sistem de calcul HPC de tip cluster
- un algoritm de distribuție dinamică a sarcinii pe unitățile de procesare eterogene ale fiecărui nod.

La nivelul distribuției sarcinilor computaționale în interiorul nodului a fost implementat un algoritm care combină două tipuri de planificare:

- una predictivă, care folosește un algoritm din familia HEFT *(Heterogeneous Earliest Finish Time*) pentru a estima pe care unitate să plaseze sarcina;
- una reactivă – care poate sesiza un dezechilibru în încărcarea unităților de procesare și, ca urmare a unei asemenea sesizări, încearcă să redistribuie sarcina printr-un algoritm de tip *work stealing* (*ws*).

Combinația între cei doi algoritmi de distribuție dinamică a sarcinii computaționale este originală (nu este prezentă între algoritmii implementați în StarPU) și elimină din potențialele neajunsuri ale fiecăruia dintre cei doi algoritmi. Algoritmii din familia *Heterogeneous Earliest Finish Time* (*HEFT)* pot lăsa unități de procesare nefolosite, pentru că tind să aloce sarcina în mod preferențial pe unitățile rapide și nu evaluează efectele pe termen lung ale predicției lor. În schimb, algoritmii de tip *work stealing* (*ws)* acționează numai după producerea dezechilibrului.

Soluția propusă este originală în contextul aplicațiilor științifice în uz, și, în special, a aplicațiilor numerice de mecanica fluidelor folosind metoda Lattice Boltzman (Palabos [Tian2014], OpenLB [Fietz2012], PowerFLOW [powerflow], walLBerla [Feichtinger2014]).

La nivelul nodului eterogen, aplicațiile OpenLB și PowerFLOW se bazează pe mediul de execuție OpenMP pentru a distribui sarcina computațională. Acest mediu nu permite distribuirea dinamică, adaptivă, a sarcinii în funcție de performanța unităților de execuție; actualmente, OpenMP permite doar folosirea dispozitivelor acceleratoare de tip GPU. Mecanismele de adaptare dinamică prin transfer de sarcină între fire nu funcționează între

unități diferite. Față de aceste aplicații, soluția propusa în acest capitol are avantajul potențial că poate adapta în mod dinamic-adaptiv distribuția pe un sistem eterogen complex (incluzând, spre exemplu, dispozitive acceleratoare de tip GPU și FPGA).

La nivelul nodului eterogen, aplicația walLBerla folosește un algoritm static de redistribuire a sarcinii între unitățile CPU și unitățile GPU**.** Algoritmul distribuie sarcina proporțional cu raportul de viteză între unități, considerat constant; considerăm că, și la acest nivel, soluția propusă în această teză are avantajul că se poate adapta dacă acest raport este variabil în timp.

O contribuție proprie constă și în extinderea planificatorului de tipul *dequeue model data aware sorted* (dmdas), bazat pe un algoritm din familia *Heterogeneous Earliest Finish Time*, în vederea implementării optimizării multicriteriale. A fost adăugat și un al treilea criteriu - încărcarea cozilor - la cele două deja existente: timpul de terminare și energia consumată. Funcția ponderată a tuturor criteriilor este folosită pentru a estima unitatea pe care se distribuie sarcina curentă.

La nivelul distribuției între noduri, a fost implementat un algoritm difuziv de distribuire și redistribuire a sarcinii între noduri. La acest nivel, componentele concurente ale aplicației sunt *chare*-urile, obiectele specifice mediului CHARM++. Algoritmul este implementat ca un protocol de negociere (pe baza protocolului Contract Net - CNP) între *chare*-uri, ceea ce reprezintă un aspect original pentru un algoritm difuziv. Algoritmul permite distribuirea inițială a sarcinii și formarea rețelei de vecinătăți a fiecărui *chare*, chiar și atunci când nu se cunosc inițial nodurile disponibile și performanțele lor. Acest lucru nu se poate realiza cu planificarea statică inclusă în aplicațiile existente.

Redistribuirea sarcinii în timpul desfășurării calculelor se face printr-un algoritm de tip transfer de sarcină (*work stealing*). Algoritmul a fost suprapus peste comunicația normală între *chare*-urile vecine de la sfârșitul unei iterații. Comunicația implementează schimbul asincron al zonelor fantomă (*ghost zones*). *Chare*-ul care solicită o zonă fantomă semnalizează de fapt că a terminat calculele. Dacă partenerul de comunicație nu a terminat procesarea aferentă, el declanșează transferul, pentru a mai reduce sarcina computațională proprie. Algoritmul din metoda walLBerla declanșează redistribuirea în mod explicit, doar când se efectuează rafinarea discretizării. Planificatorul care implementează algoritmul trebuie să știe câte noduri sunt afectate de această rafinare. El calculează care sunt vecinii care pot primi sarcina ce trebuie redistribuită. Se consideră că toți vecinii sunt disponibili si că toți au aceeași viteză de procesare. Pentru transfer, planificatorul declanșează o procedură specială de redistribuire.

Algoritmul de mapare cvasi-optimală propus în acest capitol detectează dezechilibrul în încărcare indiferent de cauză, nu generează comunicație suplimentară și nici nu folosește resurse suplimentare pentru a face calcule pentru destinația și dimensiunea transferului, pe baza unor date incerte. De aceea, chiar dacă este mai puțin eficient pentru o arhitectură omogenă și o încărcare uniformă, el este în schimb mai robust și reprezintă singura soluție în cazul în care performanțele și disponibilitatea platformei sunt incerte.

La nivelul distribuției între noduri, aplicațiile Palabos [Tian2014], OpenLB [Fietz2012] și PowerFLOW [powerflow] se bazează pe un algoritm de partiționare statică a sarcinii. Față de aceste aplicații, soluția propusă de noi are următoarele avantaje:

- poate adapta distribuția la configurații cu eterogenitate între noduri (noduri cu structură diferită) și cu un număr mare de noduri. Pentru astfel de configurații, obținerea planificării statice este dificilă;
- poate adapta distribuția la schimbări dinamice în performanța nodurilor sau a rețelei (față de cele presupuse la întocmirea planificării statice);
- poate redistribui dinamic sarcina, astfel încât să permită continuarea calculelor și după defectarea unor noduri.

La nivelul distribuției între noduri a sarcinii computaționale, aplicația walLBerla implementează și un algoritm difuziv, care ia în calcul doar creșterea sarcinii computaționale la creșterea rezoluției de discretizare. Algoritmul a fost dezvoltat pentru un sistem HPC cu noduri identice (omogen), folosit exclusiv de aplicație, având performanțe constante. Algoritmul de mapare cvasi-optimală dezvoltat în acest capitol are avantajul că ia în considerare și configurații cu eterogenitate între noduri și cu performanțe dinamice, datorită partajării platformei hardware.

6. MODELAREA ȘI SIMULAREA SISTEMULUI HARDWARE–SOFTWARE

6.1. Concepte generale

Ansamblul hardware–software a fost modelat și simulat la trei niveluri de detaliere:

1. Reprezentarea software-ului la nivel de instrucțiune și a hardware-ului la nivel de structuri ale unităților de procesare. A fost simulată rularea rutinele OpenCL pe unitățile de procesare individuale cu simulatorul *cycle accurate* Multi2Sim.
2. Reprezentarea software-ului la nivel de bloc secvențial de instrucțiuni și a hardware -ului la nivel de componente ale nodului eterogen (unități de procesare, memorie, magistrală etc) și ale rețelei de interconectare. A fost simulat un număr mic de noduri (16) interconectate, folosind mediile de modelare și simulare cu rețele Petri CPN Tools și Snoopy.
3. Reprezentarea software-ului la nivel de componentă alocabilă (*chare* sau task) și a hardware-ului la nivel de elemente de procesare interconectate. Au fost simulate nodurile eterogene și conexiunile de rețea dintre ele pentru un număr mare de noduri (1000 noduri), folosind simulatorul SimGrid.

Pe fiecare nivel de detaliere s-au folosit rezultatele obținute pe nivelul inferior și fiecare nivel a generat informație suplimentară pentru nivelul de abstractizare superior.

6.2. Simularea unităților de procesare folosind Multi2Sim

Pentru a obține timpii necesari pentru modelarea execuției kernel-urilor OpenCL, acestea au fost compilate pentru fiecare din cele două unități de procesare și rulate pe simulator. Simulările pentru CPU au fost efectuate folosind modelul unui procesor Intel Core i7-4930K (Ivy Bridge) cu șase nuclee. Simulările pentru unitatea GPU au fost efectuate folosind modelul unui procesor GPU AMD Radeon 7970, care implementează arhitectura Southern Islands.

Pe baza rezultatelor simulărilor, au fost optimizate rutinele; s-a determinat dimensiunea blocului pentru care fiecare arhitectură este exploatată optim. Timpii rezultați pentru procesarea acestor blocuri pe cele două unități sunt prezentați în tabelul 4.

Tabelul 4. Timpii obținuți la simularea kernelurilor pe Multi2Sim

	Coliziune	Propagare
CPU Intel	326 ms	116 ms
GPU AMD Radeon	84 ms	23 ms

6.3. Modelarea și simularea la nivel de nod și la nivel de interacțiune între noduri folosind rețele Petri colorate

6.3.1. Modelul dezvoltat

6.3.1.2. Schema generală

Figura 12 prezintă pagina principală a modelului realizat folosind varianta de rețele Petri colorate, utilizată în mediul CPN tools. Pentru a menține modularitatea modelului, care permite generarea lui automată, s-au folosit tranzițiile de substituție. Fiecare din tranzițiile de substituție corespunde unui bloc definit în SysML. Porturile definite pentru fiecare tranziție corespund cu porturile acestor blocuri.

Tranziția *CPU* modelează comportamentul unităților de procesare de tip multicore omogene cu nuclee complexe, nespecializate (destinate, deci, procesărilor de uz general) și, respectiv, interacțiunile cu celelalte componente ale sistemului.

În ceea ce privește interacțiunea hardware–software, tranziția *CPU* modelează execuția pentru elementele de cod, care sunt principalele expresii ale concurenței existente la nivelul aplicației. Pentru a descrie implicarea unităților de procesare de tip CPU în rularea porțiunilor de cod cu rol de suport, tranziția expune locația *CPU IDLE*, care reprezintă rezerva de nuclee CPU ca port de intrare-ieșire. Fiecare din tranzițiile care au nevoie, în acest scop, de nuclee CPU se leagă la această rezervă prin arce cu dublu sens. În această situație sunt tranzițiile *SCHEDULER, DSM, CPU DRIVER* și *GPU DRIVER*.

Declanșarea procesării este modelată prin mecanismul de declanșare a tranzițiilor. Tranziția nu se declanșează decât atunci când jetoanele care reprezintă codul și unitatea de prelucrare sunt simultan disponibile, adică simultan prezente în locațiile de intrare *CPU IDLE* și, respectiv, *THREAD START*.

Tranziția *GPU* modelează comportamentul unităților de procesare de tip manycore heterogen cu nuclee de accelerare a procesării simple, specializate, din plăcile grafice. Modelul încorporează și ierarhia de memorii la care unitățile de prelucrare au acces direct. Prelucrarea este declanșată de prezența jetoanelor reprezentând codul de prelucrat în locația *GPU START*. La terminarea prelucrării, jetoanele care semnalizează acest eveniment sunt plasate în locația *GPU END*. Pentru modelarea transferului de date, tranziția are un port de intrare – locația *GPU R* – și un port de ieșire – locația *GPU T*, prin care este conectată la tranziția *PCIe* (modelează magistrala PCIe). Pentru coordonarea transferurilor, tranziția este legată la locațiile DMA REQ și DMA ACK, prin care poate inițiа transferuri tip DMA (Direct Memory Access) către celelalte dispozitive legate la magistrală. Interacțiunea componentei DSM cu unitățile GPU a fost modelată prin legăturile directe dintre cele două tranziții prin intermediul locațiilor *GPU REQ* și *GPU ACK*. Legăturile abstractizează procesele prin care componenta DSM inițiază transferul prin DMA a zonelor de date din memoria globală a GPU.

Tranziția *MEMORY* modelează procesele care au loc în ierarhia de memorii dintre nucleele CPU și memoria principală atunci când se inițiază un acces la memorie. Tranziția este conectată la locațiile *MEM A* și *MEM R*, prin care schimbă, cu tranziția *CPU*, jetoanele

implicate în acest acces. Pentru modelarea transferului prin DMA, tranziția este conectată la tranzițiile *PCIe* și *DMA*. Conexiunea la tranziția *PCIe* se realizează prin porturile M PCI R și M PCI T, prin care circulă jetoanele reprezentând datele. Conexiunea la tranziția *DMA* se realizează prin locațiile *DMA REQ* și *DMA ACK*, prin care circulă jetoanele care reprezintă semnalele pentru gestionarea accesului la magistrală. De asemenea, prin intermediul locațiilor *M REQ* și *M ACK*, tranziția este legată la tranziția *DSM*. Mișcarea jetoanelor prin aceste porturi modelează inițierea transferului prin DMA a zonelor de date din memoria principală.

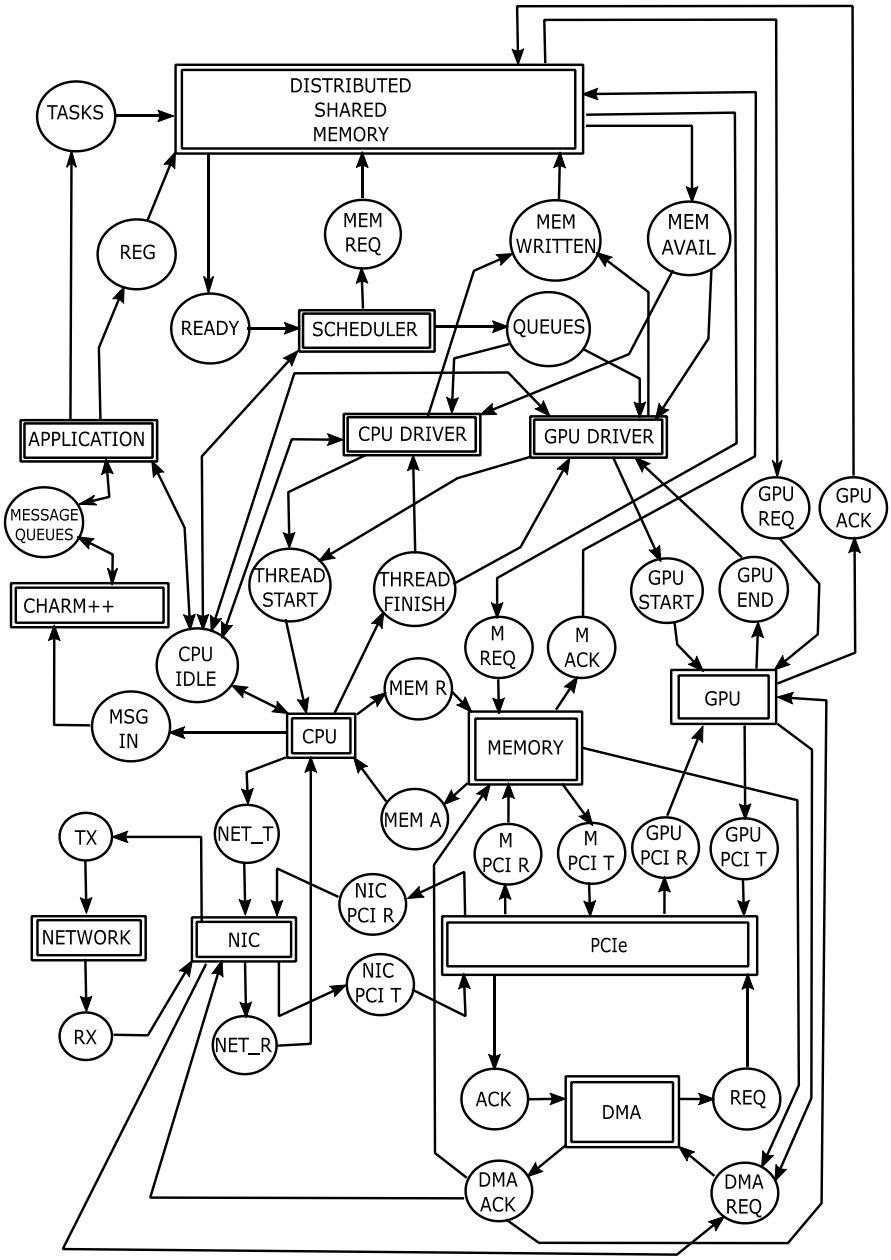


Figura 12. Vederea paginii principale a modelului

Tranziția *PCIe* modelează funcționarea magistralei PCIe, care permite transferul datelor între diferitele dispozitive ale unui nod de calcul. Tranziția se leagă prin câte un port de intrare și unul de ieșire cu fiecare din tranzițiile reprezentând aceste dispozitive. Prin aceste porturi circulă jetoane reprezentând datele transportate prin magistrală.

Tranziția *DMA* abstractizează procesele care asigură transferul de date prin magistrala PCIe între două dispozitive legate la magistrală prin mecanismul DMA, deci fără să implice

procesorul [Vin00] [Vin07] [Vin16]. Tranziția este conectată la locația *DMA REQ* (unde tranzițiile reprezentând dispozitive pot plasa jetoane cu cererile de bus) și la locația *DMA ACK*, în care plasează jetonul de sincronizare pentru dispozitivul care a primit acceptul. Tranziția interacționează cu tranziția *PCIe* prin intermediul locațiilor *REQ* și *ACK*, cu jetoane care reprezintă semnale.

Tranziția *NIC* modelează funcționarea interfețelor de rețea. Tranziția este conectată la locațiile *NET_T* și *NET_R*, prin care schimbă jetoane cu tranziția *CPU*, pentru modelarea semnalelor care coordonează transferurile prin rețeaua de interconexiune. Pentru modelarea transferului de date, tranziția este conectată la locațiile *NIC PCI T* și *NIC PCI R*, prin care jetoanele corespunzătoare sunt transferate la și de la tranziția *PCIe*. Prin locațiile *DMA REQ* și *DMA ACK* la care este conectată, tranziția *NIC* participă la modelarea arbitrajului pentru magistrala PCIe. Acest lucru permite modelarea implementărilor recente de interfețe de rețea care folosesc RDMA (*Remote DMA*).

Tranziția *NETWORK* modelează rețeaua de interconexiune care conectează nodurile între ele. Această tranziție se conectează la tranziția *NIC* prin locațiile *RX* și *TX*, prin care circulă jetoanele care reprezintă pachetele de date.

Tranziția *CHARM++* modelează coada de mesaje din mediul de execuție CHARM++. Prin locația *MSG IN*, tranziția este conectată la tranziția *CPU*, de la care primește jetoanele reprezentând mesaje. Mesajele sunt transferate prin locația *MESSAGE QUEUES* către tranziția *APPLICATION*.

Tranziția *APPLICATION* modelează aplicația propriu-zisă; tranziția preia mesajele din locația *MESSAGE QUEUES*. Jetoanele reprezentând codul care trebuie executat, ajung fie direct la unitățile de execuție prin locația *THREAD START*, fie sunt plasate în locația *TASKS*, care este punctul de intrare în infrastructura de planificare. Tranziția *APPLICATION* plasează jetoanele corespunzătoare zonelor de memorie pe care le creează, în locația *REG*. Locația *REG* alimentează tranziția *DSM*. De asemenea, tranziția *APPLICATION* este conectată la locația *CPU IDLE*, pentru a se corela execuția cu disponibilitatea unităților de prelucrare. Prin locația *THREAD STOP*, de care este legată, tranziția primește jetoanele care semnalizează terminarea execuției unui segment de cod.

Tranziția *DSM* modelează componenta de memorie virtuală distribuită a mediului StarPU. Din locația *TASKS* la care este conectată, tranziția *DSM* preia jetoanele care reprezintă task-urile create de aplicație. Tranziția plasează task-urile pregătite pentru a fi distribuite în locația *READY*. Cererile de transfer ale zonelor de memorie care provin de la tranziția *SCHEDULER* sunt preluate din locația *MEM REQ*. Jetoanele corespunzătoare zonelor de memorie transferate sunt plasate în locația *MEM AVAIL*, iar jetoanele corespunzătoare zonelor de memorie nou scrise de către tranzițiile *CPU DRIVER* și *GPU DRIVER* sunt preluate din locația *MEM WRITTEN*. Jetoanele care conțin comenzile de transfer de memorie sunt plasate în locațiile de intrare corespunzătoare ale tranzițiilor care reprezintă dispozitivele de la care se face transferul.

Tranziția *SCHEDULER* modelează componenta care planifică distribuția task-urilor pe unități de procesare. Ea preia jetoanele reprezentând task-urile pregătite pentru planificare din locația *READY*. Jetoanele cu cereri de transfer de memorie sunt plasate în locația *MEM REQ*, iar task-urile planificate în locația *QUEUE*.

Tranziția *CPU DRIVER* modelează driverul care asigură executarea task-urilor pe nucleele CPU. Ea preia jetoanele cu task-urile care sunt alocate pe nuclee CPU din locația *QUEUE* și jetoanele care conțin zonele de memorie disponibile (transferate deja) din locația *MEM AVAIL*. Tranziția *CPU DRIVER* plasează jetoanele cu task-urile de executat în locația *THREAD START*. Tranziția este conectată la locația *THREAD FINISH*, de unde preia jetoanele care semnalizează terminarea execuției task-urilor. Jetoanele care conțin zonele de date scrise la terminarea execuției task-urilor sunt transmise la tranziția *DSM* prin locația *MEM WRITTEN*.

Tranziția *GPU DRIVER* modelează driverul care asigură executarea task-urilor pe procesoarele CPU. Precum tranziția *CPU*, aceasta preia jetoanele cu task-urile care sunt alocate pe nuclee GPU din locația *QUEUE* și jetoanele care conțin zonele de memorie disponibile (transferate deja) din locația *MEM AVAIL*. Pentru porțiunea de aducere și compilare a codului GPU folosind nuclee CPU, tranziția *GPU DRIVER* plasează jetoane în locația *THREAD START*; tranziția preia jetoanele care marchează sfârșitul încărcării GPU din locația *THREAD FINISH*. Jetoanele cu task-urile de prelucrat sunt plasate în locația *GPU START*, iar jetoanele care marchează sfârșitul execuției task-urilor sunt preluate din locația *GPU END*. Jetoanele care conțin zonele de date scrise la terminarea execuției task-urilor sunt transmise la tranziția *DSM* prin locația *MEM WRITTEN*.

Tranzițiile de substituție au fost dezvoltate în subpagini. În cadrul tezei sunt prezentate descrieri detaliate ale structurii și funcționalității pentru fiecare dintre ele.

6.3.2. Simularea

6.3.2.2. Cazul distribuției statice între noduri

Au fost testate posibilitățile modelului de a reprezenta interacțiunile *chare*-urilor. Pentru a testa capabilitățile modelului dezvoltat de autor și prezentat în secțiunea 6.3.1, s-a selectat următoarea problemă de optimizare pentru mapare: Fiind dată o configurație hardware și un domeniu computațional care se potrivește din punct de vedere global pe această configurație (adică timpul de procesare vs. timpul de comunicare sunt echilibrate), care este maparea care asigură cel mai scurt timp de rulare?

Simularea a fost efectuată pentru diferite numere de *chare*-uri egal distribuite pe noduri. Aceeași simulare a fost efectuată și pe simulatorul BigSim. Rezultatele simulării sunt prezentate în Figura 28.

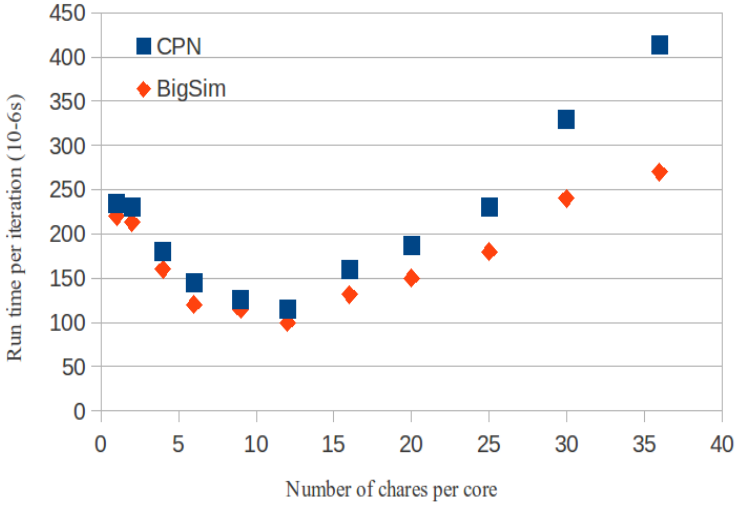


Figura 28. Comparația între simulatoarele CPN Tools și BigSim ale modelului dezvoltat [Mironescu2013]

O soluție optimă poare fi identificată între 12 și 16 *chare*-uri pe nucleu. Pentru a explica acest comportament, trebuie ținut cont de faptul că dimensiunea mesajelor (adică a zonelor fantomă ale blocurilor) scade prin descreșterea dimensiunii blocurilor. În acest fel, se reduce întârzierea până la declanșarea calculelor pentru fiecare *chare*. Explicația a fost verificată prin examinarea timpului înregistrat pentru *chare*-urile individuale. S-a observat că *chare*-urile primesc mai repede mesajele care declanșează noul calcul. În același timp, crește numărul de mesaje, astfel încât cozile de mesaje devin punctele de gâtuire. După o anumită dimensiune, mesajele mici sunt transmise mai ineficient prin rețea.

6.3.2.3. Cazul distribuției dinamice la nivel de nod

Au fost efectuate simulări pentru validarea combinației dintre modelul hardware și cel software pentru o arhitectură de procesare eterogenă multi/manycore parametrizabilă și pentru testarea modelelor de planificatoare (schedulers). Valorile teoretice au fost calculate folosind frecvența de ceas și numărul de nuclee. Ele au fost folosite pentru a caracteriza unitățile de procesare ca "rapide" (GPU) și "lente" (CPU) și raportul dintre vitezele lor este folosit în planificare. Tabelul 5 prezintă sintetic valorile astfel determinate.

Tabelul 5. Valorile determinate ale întârzierilor [Mironescu2014]

Categorie	Parametru		Valoare
Capacitate de procesare aritmetică	CPU	Teoretic	118,3 GFLOPS
		Real	68,0 GFLOPS
	GPU	Teoretic	737,4 GFLOPS
		Real	624,0 GFLOPS
Subsistem de memorie	CPU	Bandă de transfer	3 GB/s
		Memoria principală	3 GB/s
		Latență	100 ns
	GPU	Bandă de transfer	1,5 GB/s
		Memorie globală	21Gb/s
		Latență	100 ns

Pentru a pune la punct metodologia de selecție a planificatoarelor și de acordare a parametrilor lor, am considerat următoarele configurații:

[1] Configurație cu o iterație – bariera de sincronizare este la sfârșitul fiecărei iterații, pentru a permite faza de schimb între noduri prin rețeaua de comunicare. În acest caz, numărul de task-uri este egal cu n_p – numărul de partiții ale domeniului, considerând că nu există dependențe între task-uri;

[2] Iterații multiple – în anumite situații, pentru a asigura suficient paralelism (în mod special atunci când se utilizează GPU) și pentru a compensa latențele în comunicații, se fac iterații locale multiple, înainte de sincronizarea între noduri, folosind tehnica trapezului, pentru a selecta marginile domeniului curent de calcul [Meng2009]. În acest caz, dacă numărul de iterații până la sincronizare este n_i, vom avea n_i x n_p task-uri. Începând cu cea de a doua iterație, fiecare task depinde de taskul care a calculat pe aceleași date în iterația anterioară și de vecinii de șablon numeric.

Rezultatele rulărilor și simulărilor sunt prezentate în Figurile 29, 30, 31 și 32.

One iteration

Partiții egale (Figura 29)

Pentru măsurătorile reale, familia de planificatoare pe bază de liste dm (*dequeue model*) are cele mai bune performanțe. Aceste planificatoare folosesc în mod adaptiv măsurători on-line de performanță.

Partiții inegale (Figura 30)

dmda (*dequeue model data aware*) se comportă mai bine. Fiecare task este dirijat acolo unde timpul total de procesare estimat de planificator (transfer + execuție) este minim.

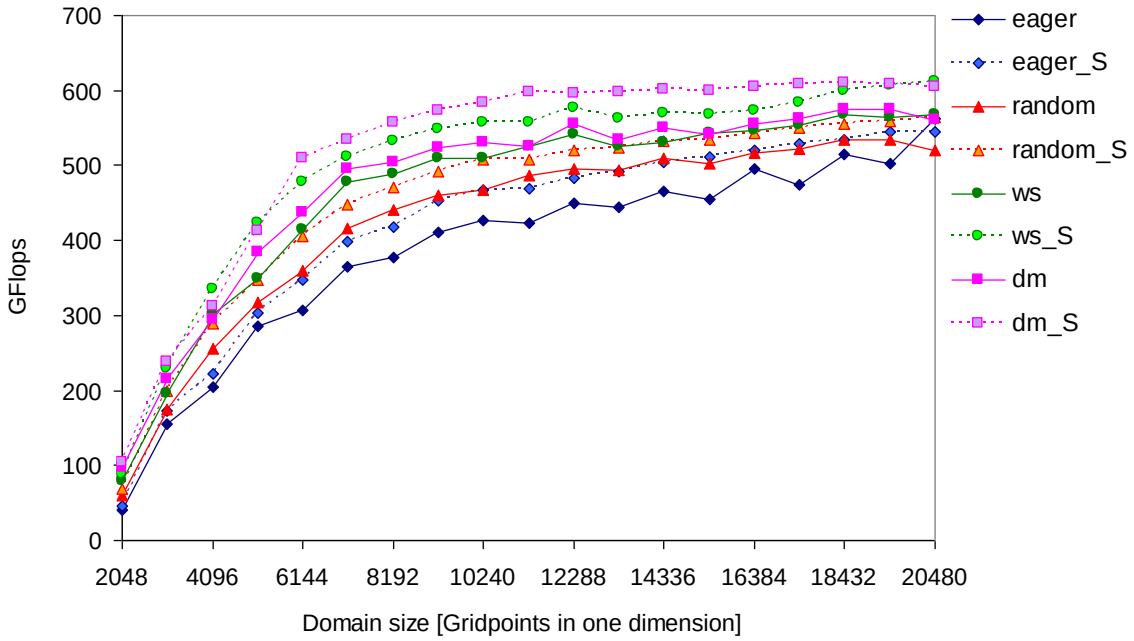


Figura 29. Rezultatele măsurătorilor și ale simulărilor (_S) pentru o iterație pe partiții egale [Mironescu2014]

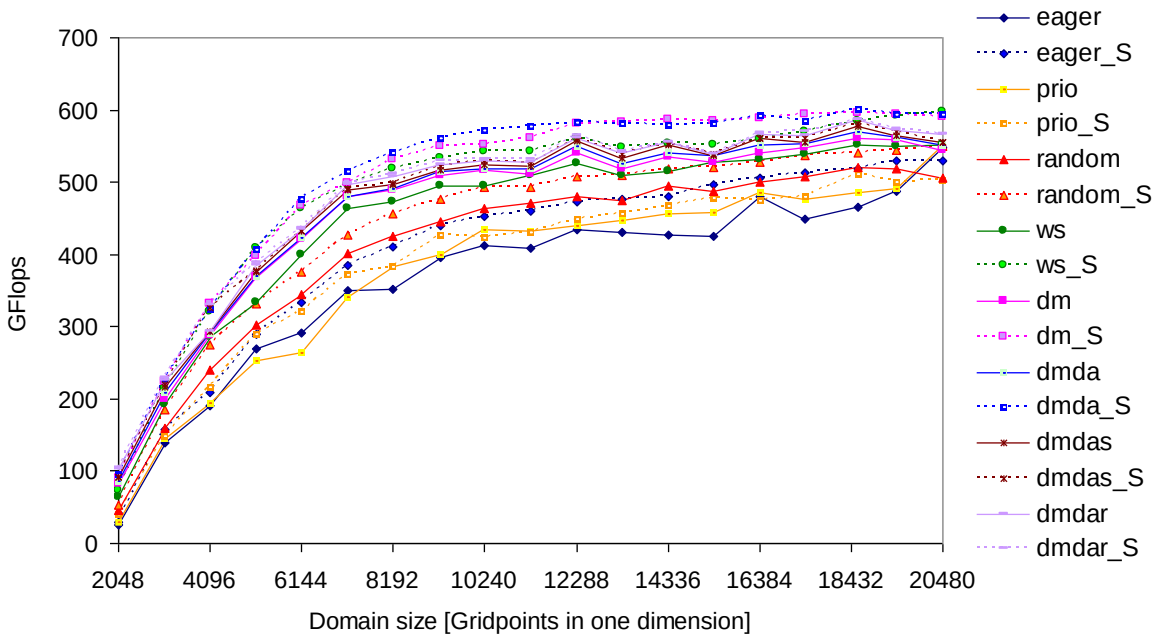


Figura 30. Rezultatele măsurătorilor și ale simulărilor (_S) pentru o iterație cu partiție inegală [Mironescu2014]

Iterații multiple

Partiție egală (Figura 31)

Planificatoarele bazate pe funcții obiectiv (de cost) folosesc mai bine resursele pentru că ele tratează mai bine atât planificarea, cât și dependențele. Planificatorul dm îmbunătățește doar distribuția task-urilor, reducând timpul de inactivitate. Schedulerul dmda planifică task-uri care procesează aceeași partiție de date preferențial pe aceeași unitate (pentru a evita transferuri costisitoare de date). Dmdas și dmdar nu au avantaje pentru partiția egală.

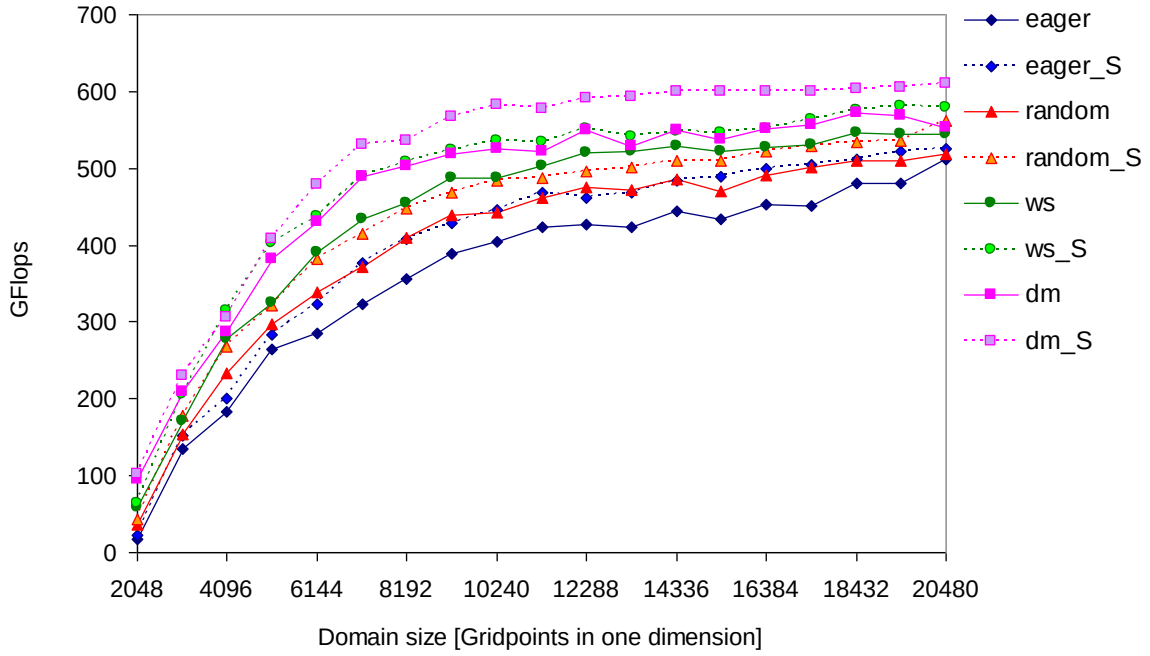


Figura 31. Rezultatele măsurătorilor și ale simulărilor (_S) pentru mai multe iterații, partiții egale [Mironescu2014]

Partiții inegale (Figura 32)

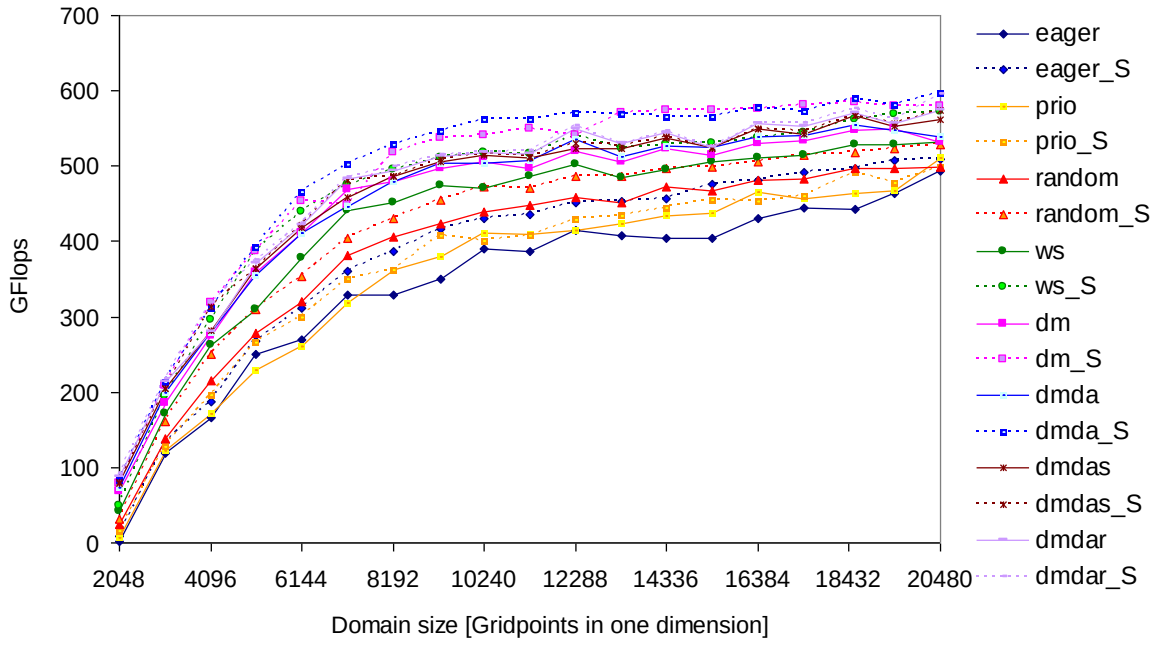


Figura 32. Rezultatele măsurătorilor și ale simulărilor (_S) pentru mai multe iterații partiții inegale [Mironescu2014]

Prioritizarea task-urilor mici folosește mai bine prefetching-ul în comparație cu cazul partițiilor egale, astfel încât planificatorul dmdas are o performanță mai bună. Planificatorul dmdar îmbunătățește planificarea pe GPU. Se reduce timpul de așteptare pentru task-urile pentru care transferul s-a terminat.

Simulările arată că modelul construit reproduce calitativ simulările reale, ceea ce îl face un instrument util, în special în faza de pre-proiectare, pentru selecția și testarea algoritmilor de planificare. Interactivitatea simulării și intuitivitatea vizuală a modelului permit sesizarea mai facilă a aspectelor care duc la scăderea performanței. De asemenea, simulările au demonstrat adecvanța planificatorilor din familia *dequeue model* (*dm*), care implementează algoritmi de tip *Heterogenous Early Finish Time* (HEFT), pentru planificarea task-urilor pe sistemul heterogen, evidențiind particularitățile fiecăruia, dar și versatilitatea planificatorului *ws*. Acest lucru a determinat alegerea pentru planificatorul pe două niveluri (capitolul 5) a unei combinații a acestor planificatori și aplicarea principiului ws și la nivelul superior.

Demersul simulării tuturor planificatoarelor implementate în StarPU cu rețele Petri era, la momentul publicării rezultatelor în [Mironescu2014], după știința noastră, singular.

6.4. Modelarea și simularea la nivel de nod și la nivel de interacțiune între noduri folosind rețele Petri extinse

6.4.1. Modelul dezvoltat

A fost folosită o abordare ierarhică pentru a asigura modularitatea modelului. Componentele principale ale sistemului au fost reprezentate ca macro-tranziții. Locațiile de conectare dintre tranziții definesc interfețele dintre componente. Tranzițiile sunt apoi rafinate în subrețele, care au locațiile de conectare ca locații de intrare și de ieșire. Figura 33 prezintă nivelul superior al ierarhiei rețelei Petri.

Componentele hardware, modelate ca macro-tranziții, sunt:

- cele două clase de unități de procesare considerate (CPU, GPU);
- sistemul de memorie;
- sistemul de comunicații (placa de rețea, rețeaua).

Fiecare macro tranziție este definită într-o subrețea care implementează comportamentul componentei.

În partea de hardware a modelului, jetoanele reprezintă resurse multiplicative – unități de procesare individuale, memorii, interfețe de rețea; acestea sunt necesare pentru efectuarea operațiilor de calcul, de acces la memorie sau de transfer prin rețea.

Modelul hardware este asamblat dintr-un număr de noduri individuale și din rețeaua de interconectare. Fiecare nod este construit din unitățile de procesare, ierarhia de memorii, interfețele de rețea și magistrala de conexiune.

Pentru fiecare unitate de procesare, modelul include o subrețea. Fiecare tranziție din subrețea modelează prelucrarea unei secțiuni de cod a aplicației (de exemplu, o rutină OpenCL). Activarea unei tranziții modelează procesarea rutinei corespunzătoare.

Fiecare declanșare a tranziției modelează execuția secvenței de cod asociate, pentru un set de date de dimensiune fixă. Se exprimă:

- timpul de procesare pentru această sarcină computațională;
- energia consumată;

- creșterea de temperatură

ca funcții de frecvența de ceas a unității. Aceste funcții pot fi determinate prin modelare cu un simulator *cycle accurate* sau prin folosirea de măsurători pe ansambluri hardware software. În capitolul 6.4. Simulare este prezentat câte un exemplu din fiecare caz.

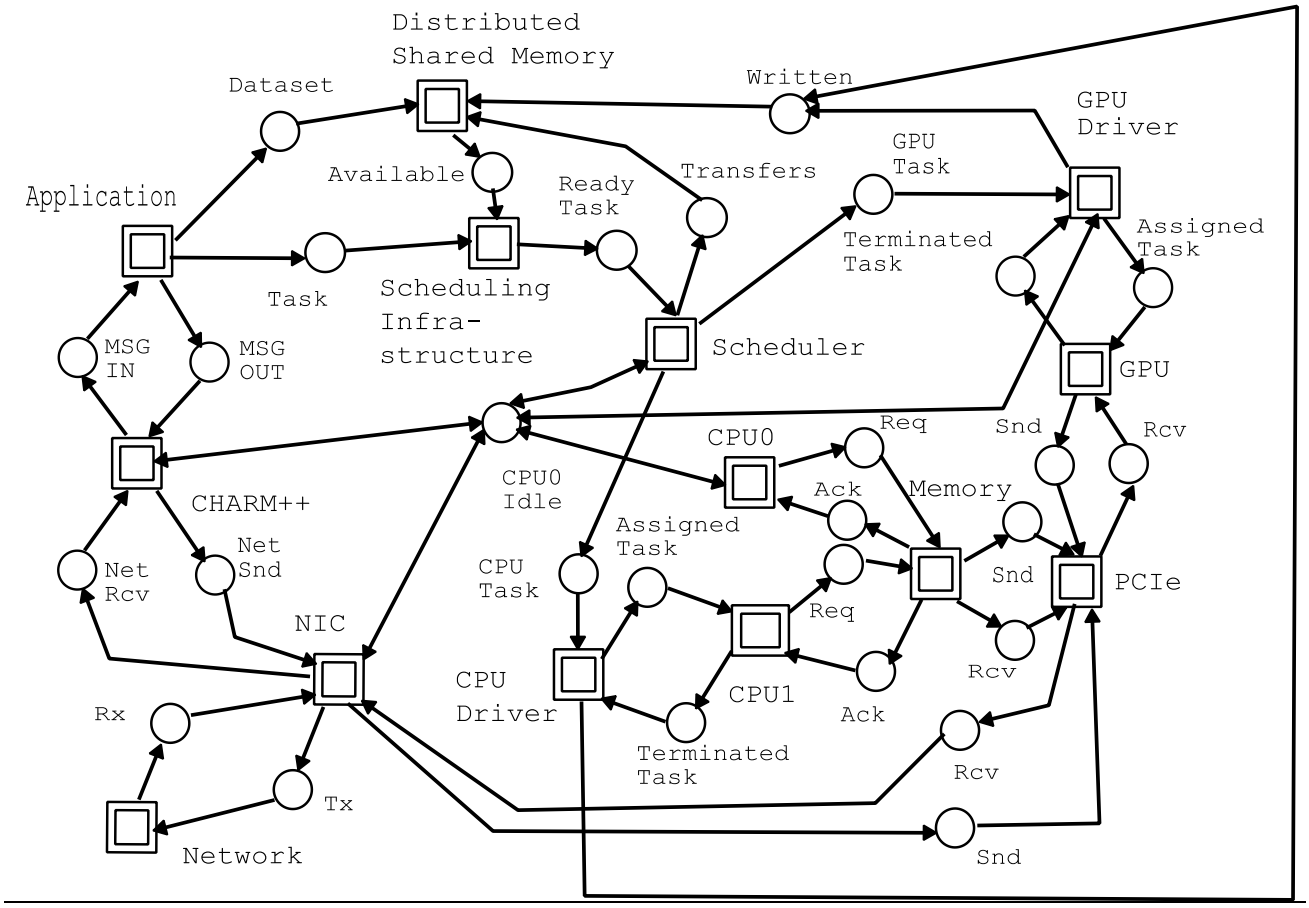


Figura 33. Planul general al rețelei Petri [adaptată după Mironescu 2017]

Subrețeaua *PCIe* care modelează magistrala PCIe implementează componenta de comunicație a modelului LogGP [Alexandrov1995]. Folosirea acestui model pentru comunicația prin PCIe a fost adaptată după [vanWerkhoven2014]. Parametrii modelului sunt folosiți ca timpi de întârziere a declanșării în tranzițiile corespunzătoare.

Subrețelele *NIC* si *Network* care modelează sistemul de comunicație implementează componenta de comunicație a modelului LoOgGP [Martinez2009]. Parametrii modelului sunt folosiți ca timpi de întârziere a declanșării în tranzițiile corespunzătoare.

Componentele software, modelate ca macro-tranziții, sunt:

- coada de mesaje (Charm++);
- managerul de memorie *Distributed Shared Memory* (StarPU);
- infrastructura de planificare (StarPU);
- driver-ul pentru fiecare PU (StarPU);
- aplicația propriu-zisă;

În partea de software a modelului, jetoanele care se deplasează prin rețea reprezintă instanțe ale structurilor de date – *chare*-uri, task-uri, mesaje, zone de date – care sunt create și actualizate.

Subrețeaua care modelează mediu CHARM++ plasează jetoanele mesaj care ajung prin sistemul de comunicații în coada de mesaje de pe nodul care conține *chare*-ul destinatar. A fost dezvoltată o structură specială, pentru că Snoopy nu are listele ca structuri de date.

Subrețeaua *Distributed Shared Memory* modelează componenta care gestionează transferul zonelor de memorie între memoria principală și memoria unității de procesare.Când planificatorul distribuie un task unui GPU , *Distributed Shared Memory* inițiază transferul jetonului Data Buffer din locația *Main memory* în locația reprezentând memoria asociată cu unitatea de procesare respectivă.

Subrețeaua *Scheduling Infrastructure* modelează infrastructura de planificare din biblioteca StarPU. Jetonele care modelează sarcina computațională a unui task parcurg locațiile corespunzătoare etapelor planificării. Subrețeaua este cuplată cu subrețeaua planificatorului selectat, care este parte a aplicației.

Pentru fiecare tip de unitate de procesare, este definită o subrețea care modelează driver-ul PU pentru acea unitate. Fiecare rețea este conectată la ieșirile corespunzătoare ale planificatorului și ale unității de procesare pe care o gestionează.

Aplicația este compusă din:

- distribuitorul de sarcină implementat în Charm++;
- planificatorul implementat în StarPU;
- kernelurile pentru calculul efectiv pe unitățile de procesare.

O reprezentare simplificată a distribuitorului de sarcină este prezentată în Figura 35. Un jeton care reprezintă un *chare* se deplasează prin subrețeaua care modelează comportamentul *chare*-ului.

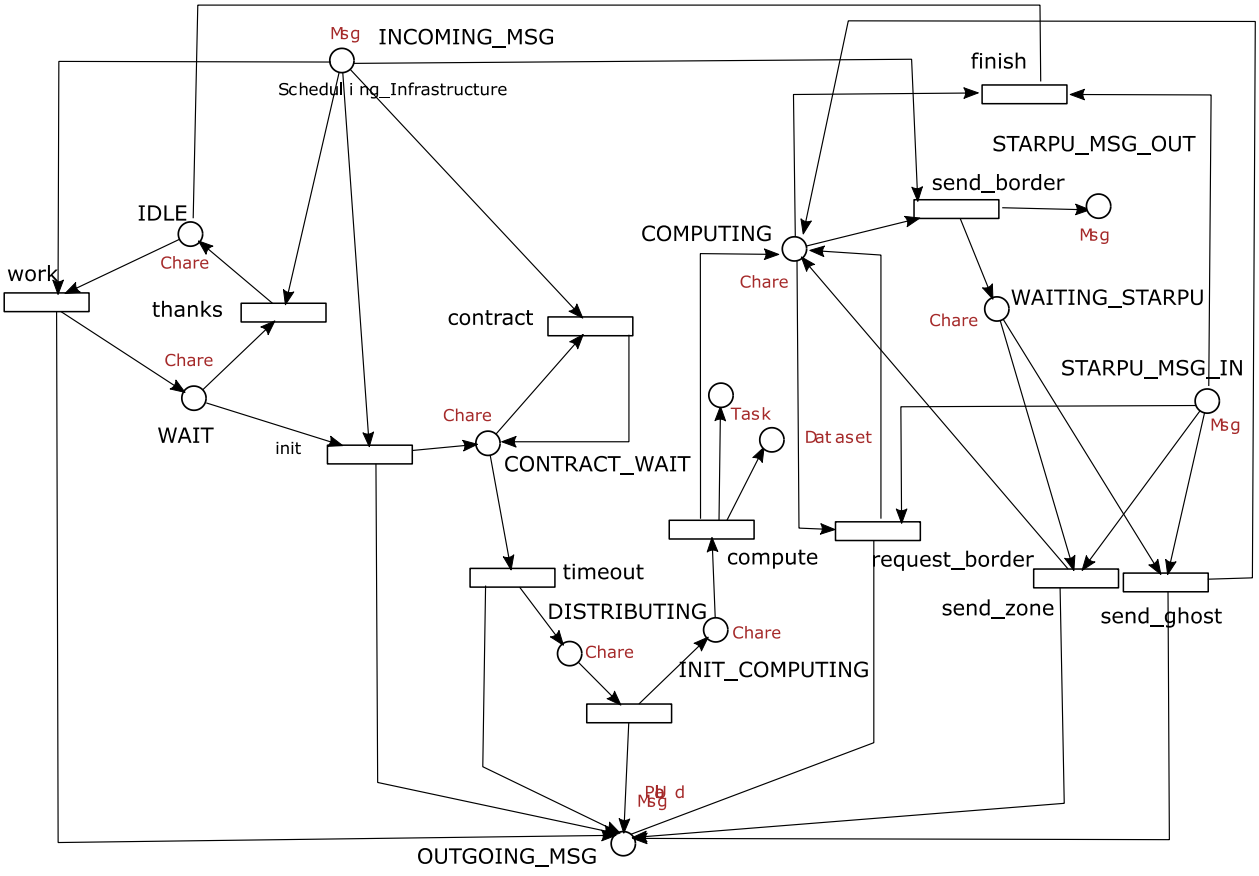


Figura 35. Modelul aplicației implementat în Snoopy [Mironescu 2017]

Subrețeaua conține câte o ramură pentru fiecare metodă accesibilă (entry method) definită de *chare*. Locația de la capătul unei ramuri este starea în care se găsește *chare*-ul după executarea metodei. Fiecare stare determină la ce mesaje poate răspunde *chare*-ul – deci, ce metode poate activa din acea stare. Ca urmare, din fiecare locație nu pleacă decât ramurile metodelor.

Punctul de intrare pentru fiecare metodă este o tranziție care se activează doar dacă *chare*-ul căruia îi este adresat se găsește în locația corespunzătoare stării în care *chare-ul* răspunde la acel mesaj și în locația corespunzătoare metodei se găsește un jeton de mesaj. Aceasta înseamnă că *chare*-ul a terminat procesarea metodei anterioare și se găsește în faza de așteptare, iar mesajul este următorul în coada de procesare.

Detaliile modelului planificatorului sunt prezentate în figura 36. Pentru fiecare task care ajunge la intrarea planificatorului, se calculează câte o funcție obiectiv pentru fiecare unitate de procesare. Task-ul va fi distribuit pe unitatea pentru care valoarea acestei funcții este minimă. În exemplul prezentat, se folosește o sumă ponderată a trei criterii de optimizare: timpul de terminare, energia consumata și încărcarea cozilor.

Modelul planificatorului a fost implementat direct în rețele simple (necolorate).

Pentru fiecare unitate de procesare, s-a construit o rețea de calcul care se activează atunci când în locația ei de intrare se plasează sarcina computațională a task-ului curent

Pe baza sarcinii computaționale aflate în locația de intrare, se generează numărul de jetoane corespunzătoare timpului de rulare al task-ului pe unitatea de procesare și se transferă într-o locație de estimare pentru timpul total. Tot aici sunt descărcate, printr-un arc de citire, jetoanele corespunzătoare valorii curente a timpului de terminare. Aceasta corespunde estimării timpului curent de terminare a task-ului curent pe unitatea de procesare.

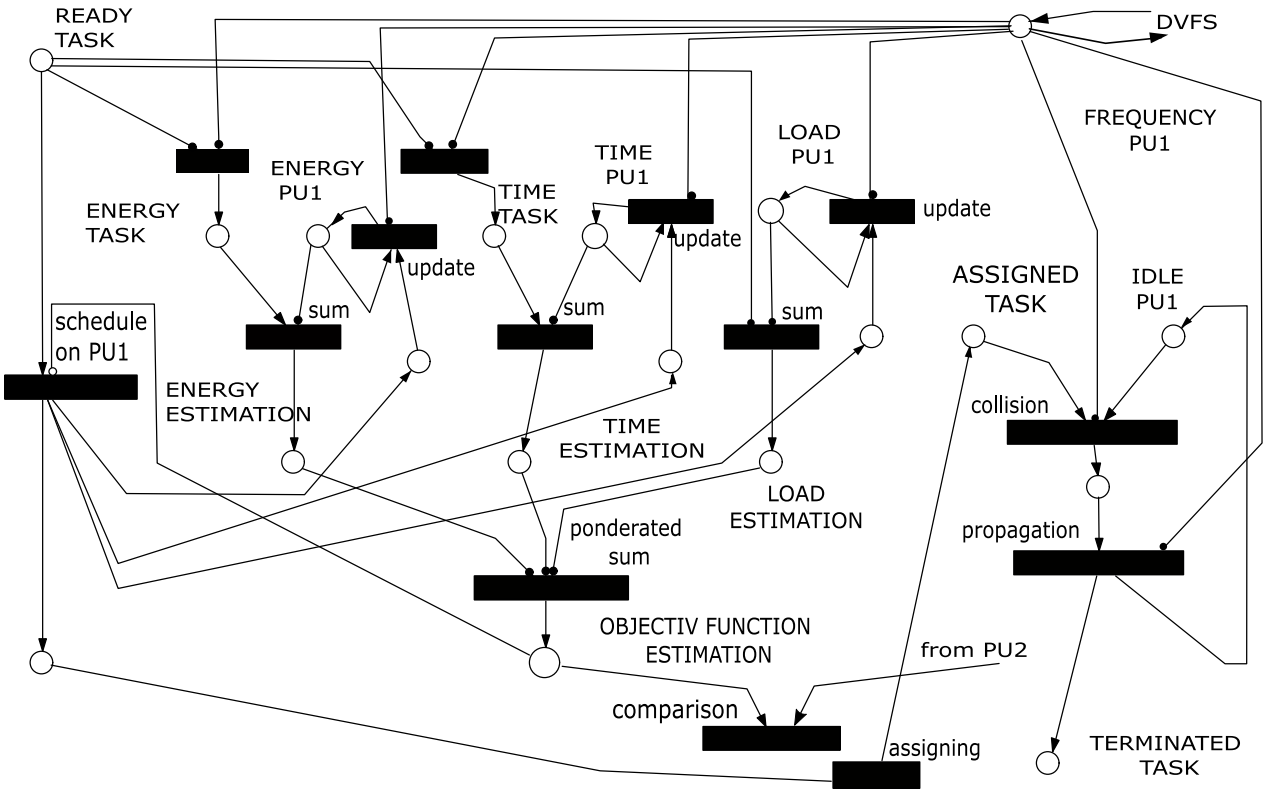


Figura 36. Modelul planificatorului implementat în Snoopy [Mironescu2018]

Asamblarea modelului

Un model particular este asamblat din componentele descrise anterior, pe baza unei descrieri structurate a hardware-ului, software-ului și mapării. Această descriere specifică numărul de noduri și, pentru fiecare nod, numărul și tipul unităților de procesare și organizarea ierarhiei de memorii. Din această descriere, prin transformare text-la-text se obține o descriere în format *(Coloured) Abstract Net Description Language* (C)ANDL a modelului, care poate fi încărcată în șirul de instrumente Snoopy.

Algoritmul pentru această construcție poate fi rezumat prin următorul pseudocod:

```
generate and connect spine switches
generate leaf switches and connect to spine switches

for each node in nodes
    for each PUi in PUs
        generate the PU net
        generate the corresponding last level cache
    end for
    for each PUi in PUs
        for each PUj in PUs
            if exists memory connection level between i,j
                generate memory level
        end for
    end for
    generate global memories
    generate network interface
    generate bus connecting devices and network interface
    generate Memory Manager
    generate Scheduling infrastructure
    generate Scheduler Skeleton
    generate Charm ++ message queue
    for each PU in PUs
        generate PU driver
        generate the cost function for PU (inclusive accumulators) and bind it in the scheduler
    endfor
 connect node to switch
endfor
```

6.4.2. Verificarea formală

Rețeaua Petri a fost analizată folosind uneltele companion pentru Snoopy, Charlie și Marcie. Din rețeaua Petri au fost obținute: matricea de incidență; arborele de acoperire (aproximat); graful de accesibilitate (aproximat). Pe baza lor, au fost investigate proprietățile structurale și comportamentale ale rețelei și au fost corelate cu proprietățile sistemului real. Rețeau Petri a modelului a fost modificată astfel încât să se asigure:

- lipsa tranzițiilor de intrare și de ieșire în rețeaua hardware, pentru că nu apar sau dispar resurse;
- mărginirea structurală și comportamentală în rețeaua hardware;
- respectarea conectivității între module în rețeaua software, așa încât să se schimbe și să se acceseze numai ce este specificat prin interfețe;
- reversibilitatea sistemului, astfel încât acesta să fie capabil să-și revină dintr-o defecțiune prin întoarcere la starea inițială.

Nerespectarea anumitor proprietăți în rețeaua software a indicat erori în structura componenetelor software. Aceste erori au fost corectate și rețeaua Petri care modelează software-ul a fost adaptată să reprezinte noua structură. Astfel, pentru noua rețea, s-a asigurat:

- mărginirea, ceea ce garantează gestionarea corectă a memoriei;
- viabilitatea, ceea ce garantează lipsa blocajelor (deadlocks);
- reversibilitatea, ceea ce garantează că sistemul se întoarce la starea de unde a plecat,.

O verificare formală mai riguroasă a fost făcută prin utilizarea suportului oferit pentru logica temporală liniară LTL în Charlie. Prin verificarea expresiei *AF DONE*, care afirmă că, pe orice cale, se va ajunge la un moment dat în viitor ca locația *DONE* să fie marcată, se garantează că sistemul parcurge toate etapele de calcul din algoritmul numeric.

6.4.3. Simularea

6.4.3.1. Considerații generale

În acest caz, simularea nu mai este interactivă. Trebuie precizat un număr de pași de simulare și trebuie selectat un motor de simulare.

Timpul nu mai poate fi citit din jetoane, ci trebuie luat din jurnalul simulării. Pentru a pregăti simularea, trebuie specificați timpii de întârziere pe tranziții și marcajul inițial al rețelei.

Configurarea automată este făcută prin intermediul transformărilor M2T, care transformă modelul sistemului în rețeaua pregătită pentru simulare.

6.4.3.2. Cazul distribuției dinamice

Simularea rețelei finale (X)GSPNc a fost efectuată cu instrumentul Snoopy, pentru a aprecia dacă soluția propusă se comportă într-adevăr mai bine decât o soluție care nu folosește echilibrarea sarcinii computaționale. Simulările au fost efectuate cu trei rețele:

1) Fără echilibrare, folosind doar sincronizare în faza de transfer și distribuție statică (modelul de referință);
2) Cu redistribuirea sarcinii numai la nivelul internoduri și partiție egală;
3) Cu redistribuirea sarcinii pe ambele niveluri.

Rețeaua cu redistribuire completă a reușit o reducere a timpului de calcul cu 15%, în comparație cu rețeaua fără echilibrare. Rețeaua cu redistribuire locală (Net no. 2) a reușit o reducere de 8,6% a timpului de calcul, comparativ cu modelul de referință.

6.4.3.3. Cazul optimizării dinamice multicriteriale

6.4.3.3.1. Sistemul de calcul modelat

Am multiplicat celula de bază a clusterului COKA (“Computing On Kepler Architectures”) de la Universitatea din Ferrara [coka] de patru ori, obținând arhitectura prezentată în Figura 37. Nodurile au o înălțime de 4 unități de rack U (rack unit), așa că un rack găzduiește două celule de bază și folosește un singur switch IB frunză. Cele două rack-uri sunt conectate printr-un switch IB de trunchi. Pentru a crește gradul de relevanță al cazului studiat, fiecare nod a fost configurat diferit de celelalte, în ceea ce privește CPU-urile și GPU-urile disponibile.

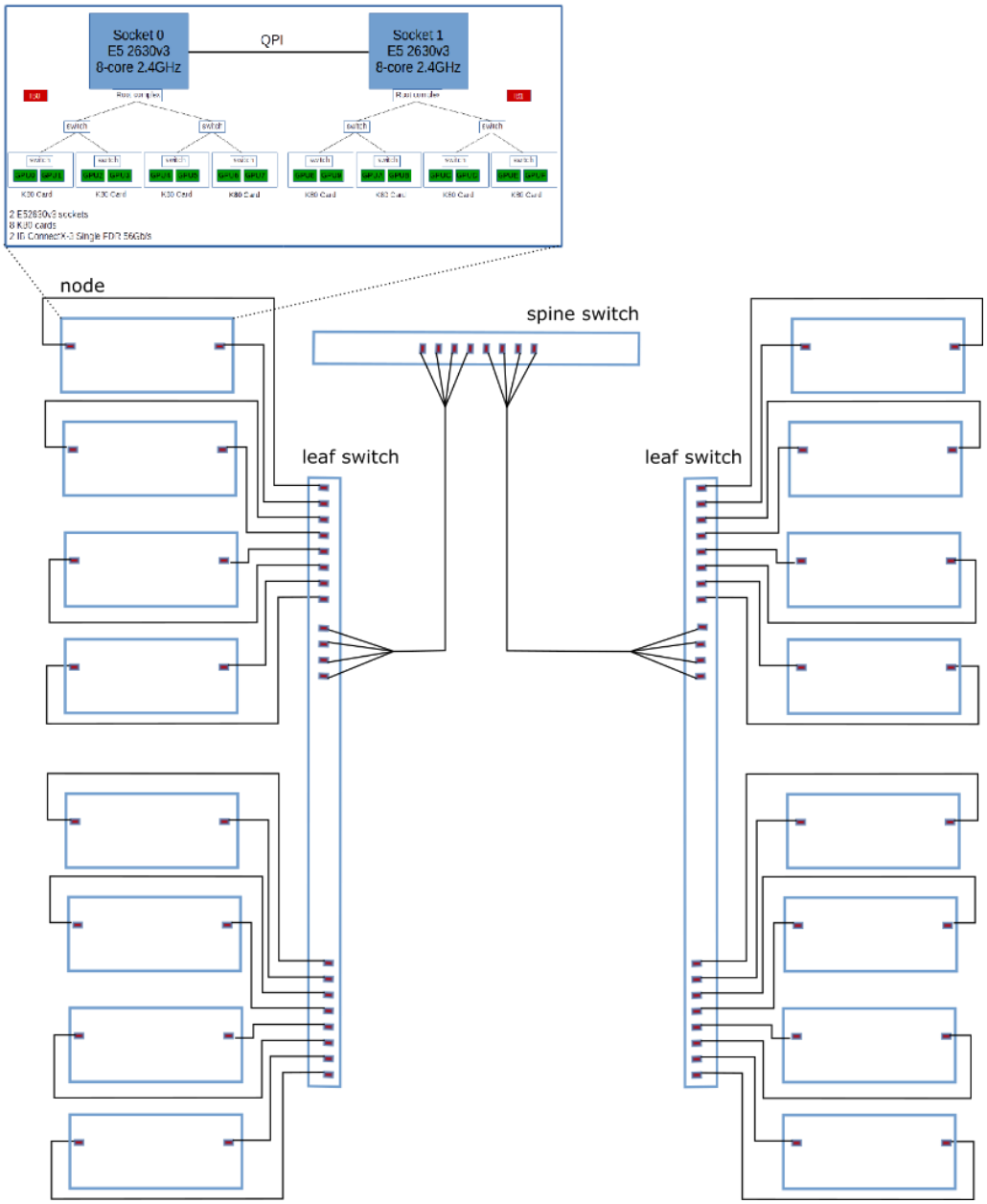


Figura 37. Arhitectura HPC modelată

Configurația pentru fiecare nod este dată în Tabelul 6, unde:

- NBO (i) este numărul de blocuri care ocupă complet unitățile nodului i;
- nCPU (i) și nGPU(i) se referă la numărul de PU ale fiecărui tip.

Fiecare CPU are opt nuclee și fiecare card GPU are două procesoare.

Tabelul 6. Configurația nodurilor

Nodul i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
nCPU (i)	8	8	8	8	8	8	8	8	16	16	16	16	16	16	16	16
nGPU (i)	2	4	6	8	10	12	14	16	2	4	6	8	10	12	14	16
NBO (i)	23	38	53	68	83	98	113	128	31	46	61	76	91	106	121	136

6.4.3.3.2. Spațiul variantelor de proiectare

Tabelul 7 prezintă spațiul soluțiilor de proiectare pentru cazurile considerate.

Tabelul 7. Spațiul soluțiilor de proiectare pentru cazurile considerate

Parametru	Inițial	Bloc	"Redus"
Frecvența CPU	13	4	4
Frecvența GPU	25	12	12
Unitate de partiție	Punct de discretizare	Bloc (1024x1024 puncte)	Unitate=16 blocuri
Număr de unități de partiție	~10^9	256	16
Număr de noduri	16	16	16
Cardinalitate	$(13*25)^{16}$ * P(16)*$16^{10\wedge 9}$ = $1{,}5*10^{40}$ *10^8*$16^{10\wedge 9}$	48^{16}* P(16)*16240	48^{16}*P(16) = 10^{27}*10^8

6.4.3.3.3. Modele matematice

Execuția aplicației HPC pe arhitectura hardware parametrizabilă considerată a fost modelată ca o combinație de operații atomice de calcul și de transfer care pot fi efectuate pe unitățile funcționale existente.

Calculul este împărțit în două faze: coliziune și propagare (capitolul 4, pagina 51).

Un nucleu al unui CPU procesează în mod optim (din punctul de vedere al acceselor la memorie) un bloc cu 1024x1024 de puncte. Timpul de calcul pentru acest număr de puncte în faza de coliziune (t col_CPU) depinde de frecvența de ceas a unității (f, în MHz) prin formula:

$$t_{col\_CPU} = \frac{1380}{f} \text{ [s]} \quad (3)$$

Energia folosită în această fază pentru calculul aceluiași număr de puncte (E_{col_CPU}) depinde de frecvența de ceas a unității (f, în MHz) prin ecuația:

$$E_{col\_CPU} = P_{col\_CPU} \cdot t_{col\_CPU} \quad (4)$$

unde puterea consumată în faza de coliziune (P_{col_CPU}) este dată de ecuația:

$$P_{col\_CPU} = 30 + 0{,}0062 \cdot f^{1+0{,}2} \quad (5)$$

Pentru faza de propagare, timpul de calcul pentru același bloc (tprop_CPU) este independent de frecvență:

$$t_{prop\_CPU} = 0{,}15\ s \quad (6)$$

Energia folosită de CPU în faza de propagare (E_{prop_CPU}) depinde de frecvență (f, în MHz) prin relația:

$$E_{prop\_CPU} = P_{prop\_CPU} \cdot t_{prop\_CPU} \text{ [J]} \quad (7)$$

unde puterea consumată în faza de propagare (P_{prop_CPU}) este dată de ecuația:

$$P_{prop\_CPU} = 30 + 0{,}055 \cdot f^{1+0{,}2} \text{ [W]} \quad (8)$$

Un procesor GPU calculează, în mod optim, un bloc de 8192x1024 puncte. Timpul necesar pentru calculul acestui bloc pe GPU în faza de coliziune (t col_GPU) depinde de frecvența de ceas a unității (f, în MHz) prin formula:

$$t_{col\_GPU} = \begin{cases} \frac{71{,}4}{f} & , \quad f < 650\ MHz \\ 0{,}033 & , \quad f \geq 650\ MHz \end{cases} \quad (9)$$

Energia folosită în această fază (Ecol_GPU) este dată de formulele:

$$E_{col\_GPU} = P_{col\_GPU} \cdot t_{col\_GPU} \quad (10)$$

$$P_{col\_GPU} = \begin{cases} 42{,}5 + 0{,}109 \cdot f & , \quad f < 650\ MHz \\ 42{,}5 + 0{,}109 \cdot f + 0{,}005 \cdot e^{0{,}0099 \cdot f} & , \quad f \geq 650\ MHz \end{cases} \quad (11)$$

unde P_{col_GPU} este puterea necesară pentru în faza de coliziune pe unitatea GPU.

Pentru faza de propagare pe unitatea GPU (t prop_GPU), timpul necesar este:

$$t_{prop\_GPU} = \begin{cases} \frac{27{,}2}{f} & , \quad f < 800\ MHz \\ 0{,}085 & , \quad f \geq 800\ MHz \end{cases} \quad (12)$$

Energia folosită pentru faza de propagare (E_{prop_GPU}) este calculată pe baza puterii electrice necesare (P_{prop_GPU}) cu formulele:

$$E_{prop\_GPU} = P_{prop\_GPU} \cdot t_{prop\_GPU} \quad (13)$$

$$P_{prop\_GPU} = 42{,}94 + 0{,}096 \cdot f \quad (14)$$

Creșterea de temperatură în GPU a fost modelată folosind modelul RC (thermal Rezistence Conductivity), model descris în [Hong2010], pe care au fost adaptate datele din [Calore2017]. Modelul folosește o dependență exponențială de timp a creșterii asimptotice a temperaturii până la o temperatură maximă (τ_{max}):

$$\tau_{rise}(t) = \tau_{idle} + (\tau_{max} - \tau_{idle}) \cdot \left(1 - e^{-\frac{t}{RC\_Rise}}\right) \quad (15)$$

Temperatura maximă τ_{max} este dependentă de puterea curentă (P_{comp}) prin expresia:

$$\tau_{max}(P_{comp}) = 0{,}15 \cdot P_{comp} + 6{,}2 + 30 \cdot MAI \quad (16)$$

unde P_{comp} este calculată cu formula (11) pentru faza de coliziune, respectiv cu relația (14) pentru faza de propagare.

Tabelul 8 prezintă valorile pentru intensitatea acceselor la memorie (MAI). MAI reprezintă procentajul de instrucțiuni cu acces la memorie din totalul de instrucțiuni procesate. Valorile au fost aproximate pe baza considerentelor, prezentate în [Calore2017], legate de predominanța acceselor la memorie asupra calculelor, funcție de faze și de frecvențe.

Tabelul 8. Valorile intensității acceselor la memorie (MAI) la calculul pe GPU

Faza	Frecvența [MHz]	MAI
Coliziune	<650	0,5
Coliziune	>650	1
Propagare	<800	0,8
Propagare	>800	1

O funcție similară este folosită pentru modelarea scăderii asimptotice a temperaturii la valoarea pentru unitatea de procesare inactivă:

$$\tau_{decay}(t) = \tau_{idle} + (\tau - \tau_{idle}) \cdot e^{-\frac{t}{RC\_Deacay}} \quad (17)$$

Acest fenomen are loc atunci când procesorul este comutat la starea inactivă. Temperatura pentru procesorul inactiv a fost considerată 40ºC.

Blocurile care trebuie procesate sunt în memoria principală. Timpul și energia pentru accesul unităților de procesare la memoria locală sunt incluse în modelele prezentate anterior. Pentru a fi procesate de GPU, blocurile trebuie transferate din memoria principală (memoria gazdei) în memoria locală a GPU-ului (memoria dispozitivului). Rezultatul trebuie, apoi, transferat înapoi din memoria dispozitivului în cea a gazdei.

Pentru a modela aceste transferuri prin magistrala PCIe, am folosit modelul descris în [vanWerkhoven2014]**.** Acest model adaptează modelul latență (L), overhead (o), short message gap (g), long message gap (G) la comunicația gazdă-dispozitive prin bus-ul PCIe. Expresia timpului de transfer de la gazdă la dispozitiv (t_{hd}) ca funcție de numărul de octeți transferați (B) este dată de formula:

$$t_{hd}(B) = L_{hd} + o_{hd} + G_{hd} \cdot B + g_{hd} \quad (18)$$

Cu valorile din [vanWerkhoven2014], expresia (18) devine:

$$t_{hd}(B) = 0.009420 + 8{,}318392E-008 \cdot B + 0.002503 \text{ [ms]} \quad (19)$$

Energia folosită pentru transferul gazdă-dispozitiv (E_{hd}) este:

$$E_{hd} = P_{PCI} \cdot t_{hd} \quad (20)$$

P_{PCI} este puterea consumată pe magistrala PCIe și este 30 W [pcie].

Timpul de transfer dispozitiv-gazdă (t_{dh}) este dat de formula:

$$t_{dh}(B) = L_{dh} + o_{dh} + G_{dh} \cdot B + g_{dh} \quad (21)$$

Cu valorile din [van Werkhoven2014], expresia (21) devine:

$$t_{dh}(B) = 0.009023 + 7{,}924734E-008 \cdot B + 0.002674 \text{ [ms]} \quad (22)$$

Energia folosită pentru transferul dispozitiv-gazdă (E_{hd}) este:

$$E_{dh} = P_{PCI} \cdot t_{dh} \quad (23)$$

Pentru transferul prin rețea a fost folosit modelul LoOgGP propus de [Martinez 2009]. Acest model introduce suplimentar un al doilea overhead (O) la parametrii latență (L), overhead (o), short message gap (g), long message gap (G). Acest overhead este liniar dependent de dimensiunea transferată. Timpul de transfer între două noduri (t_{IB}), folosind rețeaua InfiniBand (IB) pentru B octeți, este:

$$t_{IB}(B) = L_{IB} + o_{IB} + O_{IB} \cdot B + G_{IB} \cdot B + g_{hd} \quad (24)$$

Cu valorile din [Martinez2009], timpul pentru transferul prin rețeaua IB formula devine:

$$t_{IB}(B) = 181{,}5 + 34{,}7 + 1{,}88 \cdot B + 37{,}9 \cdot B + 1{,}88 \text{ [ms]} \quad (25)$$

Energia folosită pentru transfer este dată de formula:

$$E_{IB}(B) = t_{IB}(B) \cdot P_{IB} \text{ [ms]} \quad (26)$$

unde P_{IB} este puterea consumată de o legătură IB; P_{IB}=6W [ib_melanox].

Valorile parametrilor din modelul matematic au fost introduse în timpii de întârziere, expresiile de pe arce și ratele de transfer ale modelului cu rețele Petri extinse stohastice. Astfel, în modelul CPU, formula (3) a fost folosită pentru timpul de întârziere pentru tranziția corespunzătoare rutinei de coliziune și formula (6), în mod similar, pentru tranziția de propagare.

Sarcina computațională care este prelucrată la o tranziție de un nucleu CPU este cea corespunzătoare unui bloc de 1024x1024 puncte de discretizare.

Pentru modelul GPU, fiecare tranziție corespunde calculului a opt blocuri – un bloc GPU. Pentru timpii de întârziere din tranzițiile corespunzătoare, s-au folosit formulele (9) pentru coliziune și (12) pentru propagare.

Pentru inscripțiile de pe arce care specifică numărul de jetoane transferate în locația acumulator *Energy*, s-au folosit formulele (4) pentru coliziune pe CPU, (7) pentru propagare pe CPU, (10) pentru coliziune pe GPU și respectiv (13) pentru propagare pe GPU.

Locația acumulator *Temperature* este alimentată cu jetoane când nucleul este activ și golită de jetoane atunci când nucleul este inactiv, prin tranziții care implementează funcțiile modelului (15), respectiv (17).

Valorile parametrilor din formulele (19) și (24) au fost folosite în formulele care dau întârzierile tranzițiilor și ratele de transfer pentru modelul PCIe.

Rețeaua PCIe a fost configurată conform arhitecturii sistemului modelat. Ea are câte o magistrală separată pentru fiecare din cele două socluri. În acest fel, doar jumătate din plăcile grafice (în acest caz, patru cu opt procesoare) ale unui nod concurează pentru aceeași legătură gazdă-dispozitiv.

Pentru modelul rețelei, valorile parametrilor din formula (25) au fost folosite pentru întârzierile și ratele de transfer ale tranzițiilor din subrețeaua corespunzătoare.

6.4.3.3.4. Modelul de referință

Pentru a avea o referință, am construit un model matematic original care estimează timpul și energia necesare pentru efectuarea unei iterații, pentru o configurație dată a sistemului. O configurație de sistem specifică numărul N_N de noduri și, pentru fiecare nod i ($0<i<N_N$), numărul de unități de procesare din fiecare tip ($n_{CPU}(i)$ și $n_{GPU}(i)$), frecvențele de ceas ale fiecărei unități ($f_{CPU}(j,i)$, $1<j< n_{CPU}(i)$ și $f_{GPU}(k,i)$ $1<k< n_{GPU}(i)$) și numărul de blocuri alocate $N_B(i)$.

Timpul de calcul al unui bloc pe nucleul CPU j al nodului i ($t_{CPU}(j,i)$) este:

$$t_{CPU}(j,i) = t_{col\_CPU}(f_{CPU}(j,i)) + t_{prop\_CPU}(f_{CPU}(j,i)) \quad (27)$$

unde t_{col_CPU} și t_{prop_CPU} sunt calculate cu formulele (3) și respectiv (6).

Energia este:

$$E_{CPU}(j,i) = E_{col\_CPU}(f_{CPU}(j,i)) + E_{prop\_CPU}(f_{CPU}(j,i)) \quad (28)$$

unde E_{col_CPU} și E_{prop_CPU} sunt calculate cu formulele (4) și respectiv (7).

Dacă procesorul GPU folosește exclusiv legătura cu memoria gazdei, timpul total de calcul pentru opt blocuri ($B_{GPU}=8*B_{CPU}$ bytes) pe procesorul GPU k al nodului i ($t_{GPU}(k,i)$) este:

$$t_{GPU}(k,i) = t_{hd}(B_{GPU}) + t_{col\_GPU}(f_{GPU}(k,i)) + t_{prop\_GPU}(f_{GPU}(k,i)) + t_{dh}(B_{GPU}) \quad (29)$$

unde t_{hd} și t_{dh} sunt timpii de transfer de la gazdă la dispozitiv și, respectiv, de la dispozitiv la gazdă, dați de ecuațiile (18) și (21), iar t_{col_GPU} și t_{prop_GPU} sunt timpii pentru calculul în faza de coliziune, respectiv propagare, calculați cu formulele (9) și respectiv (12).

Energia folosită pentru a calcula opt blocuri pe procesorul GPU k al nodului i (ECPU j,i) este dată de formula:

$$E_{GPU}(k,i) = E_{hd}(B_{GPU}) + E_{col\_GPU}(f_{GPU}(k,i)) + E_{prop\_GPU}(f_{GPU}(k,i)) + E_{dh}(B_{GPU}) \quad (30)$$

unde energiile necesare pentru calcul Ecol_GPU și Eprop_GPU sunt calculate cu formulele (10) și (13), iar energiile pentru transport Ehd și Edh sunt calculate cu formulele (20) și (23).

Pentru a estima timpul ($t(i,N_B)$) și energia ($E(i,N_B)$) necesare pentru calculul a N_B blocuri pe nodul i, am plecat de la următoarele premise:

- planificatorul va încerca să ocupe toate unitățile disponibile;
- pe unitățile GPU se alocă unități de opt blocuri pentru utilizare optimă;
- dacă sunt mai puțin de opt blocuri disponibile pentru planificare, ele vor fi distribuite pe nucleele CPU disponibile.

Pentru cazul în care m unități GPU partajează aceeași magistrală către memorie, pentru timpul maxim pe care unitățile GPU îl necesită pentru calcul (t_{max_GPU}), am dedus formula:

$$t_{max\_GPU} = (m \bmod 2) \cdot t_{hd}(B_{GPU}) + \text{floor}(\frac{m}{2}) \cdot t_{hd}(2 \cdot B_{GPU}) + t_{col_{GPU}}(f_{GPU}(k,i)) + t_{prop_{GPU}}(f_{GPU}(k,i)) + ((m-1) \bmod 2) \cdot t_{dh}(2 \cdot B_{GPU}) + (m \bmod 2) \cdot t_{dh}(B_{GPU}) \quad (31)$$

Notăm cu:

- $N_{BO}(i)$ numărul de blocuri care ocupă complet unitățile de procesare ale nodului i;
- q câtul împărțiri N_B/ $N_{BO}(i)$;
- r restul acestei împărțiri.

Timpul necesar calculului a N_B blocuri pe nodul i este dat de ecuația:

$$t(i, N_B) = (q+1) \cdot \max_{1 \le j \le n_{CPU}(i)} (t_{CPU}(j,i), t_{max\_GPU}) \quad (32)$$

Energia consumată pentru cele N_B blocuri este dată de relația:

$$E(i, N_B) = q \cdot \left(\sum_{j=1}^{n_{CPU}(i)} E_{CPU}(j,i) + \sum_{k=1}^{n_{GPU}(i)} E_{GPU}(k,i) \right) + \sum_{k=1}^{\frac{r}{8}} E_{GPU}(k,i) + \sum_{j=1}^{r \bmod 8} E_{CPU}(j,i) + \sum_{j=1}^{n_{CPU}(i)} E_{CPUidle}(j,i) + \sum_{k=1}^{n_{GPU}(i)} E_{GPUidle}(k,i) \quad (33)$$

unde $E_{CPUidle}(j,i)$ este energia consumată de unitatea CPU j a nodului i în stare inactivă și $E_{GPUidle}(k,i)$ este energia consumată de unitatea GPU k a nodului i în stare inactivă.

Energiile în stare inactivă sunt calculate cu formulele:

$$E_{CPUidle}(j,i) = t_{CPUidle}(j,i) \cdot P_{CPUidle} \quad (34)$$

și respectiv:

$$E_{GPUidle}(j,i) = t_{GPUidle}(j,i) \cdot P_{GPUidle} \quad (35)$$

Timpul petrecut în stare inactivă este dat de ecuațiile:

$$t_{CPUidle}(j,i) = \begin{cases} t(i,N_B) - (q+1) \cdot t_{CPU}(j,i), & j \le r\ mod(8) \\ t(i,N_B) - q \cdot t_{CPU}(j,i), & j > r\ mod(8) \end{cases} \quad (36)$$

$$t_{GPUidle}(j,i) = \begin{cases} t(i,N_B) - (q+1) \cdot t_{GPU}(k,i), & k \le r \text{ div } 8 \\ t(i,N_B) - q \cdot t_{GPU}(k,i), & k > r\ div\ 8 \end{cases} \quad (37)$$

Am considerat termenii liberi din ecuațiile (5) și (14): $P_{CPUidle}$=30 și $P_{GPUidle}$=42,5.

După ce toate unitățile și-au terminat calculele locale, are loc faza de schimb. În această fază, fiecare nod transferă zonele de margine la vecini.

Pentru că subdomeniile pot avea diferite dimensiuni și forme, am folosit o valoare medie pentru dimensiunea (în octeți) a zonei de margine (d_{bord}). Pentru un subdomeniu cu NB blocuri, această valoare este:

$$d_{bord}(N_B) = 4 \cdot SQRT(N_B) \cdot 1024 \cdot w_{bord} \cdot d_{site} \quad [\text{bytes}] \quad (38)$$

unde w_{bord} este lățimea zonei de margine (dimensiunea șablonului numeric pentru un punct) și d_{site} este dimensiunea, în octeți, a spațiului de memorie necesar pentru un punct. Medierea este realizată considerând un subdomeniu pătrat cu același număr de blocuri.

Dacă se consideră un canal duplex de comunicare, astfel încât recepția și trimiterea se pot face în paralel, timpul în care nodul i este gata pentru o nouă iterație ($t_{iter}(i)$) este suma dintre timpul necesar pentru a calcula cele N_B blocuri $t(i,N_B)$, calculat cu formula (32), și timpul de transfer al zonelor de margine t_{IB}, calculat cu formula (25):

$$t_{iter}(i, N_B) = t(i, N_B) + t_{IB}(d_{bord}(N_B)) \quad (39)$$

Energia consumată în acest timp este:

$$E_{iter}(i, N_B) = E(i, N_B) + E_{IB}(d_{bord}(N_B)) \quad (40)$$

unde $E(i,N_B)$ este energia consumată de nodul i pentru calculul a N_B blocuri, calculată cu formula (33), iar E_{IB} este energia necesară pentru transferul zonelor de margine, calculată cu formula (26).

O iterație este terminată atunci când toate nodurile și-au completat transferurile. Pentru o distribuție de blocuri $N_B(i)$ $0 \le i \le N_N$, obiectivele de optimizare pentru întregul sistem sunt:

a) timpul de calcul pentru o iterație (t_{iter}):

$$t_{iter} = \max_{0 \le i \le N_N} (t_{iter}(i, N_B(i))) \quad (41)$$

b) energia de calcul pentru o iterație(E_{iter}):

$$E_{iter} = \sum_{i=0}^{N_N - 1} E_{iter}(i, N_B) + E_{idle} \quad (42)$$

unde energia în starea inactivă E_{idle} este dată de formula:

$$E_{idle} = \sum_{i=0}^{N_N - 1} (t_{iter} - t_{iter}(i, N_B)) \cdot \left(\sum_{j=1}^{n_{CPU}(i)} P_{CPUidle}(j, i) + \sum_{k=1}^{n_{GPU}(i)} P_{GPUidle}(k, i) \right) \quad (43)$$

c) dezechilibrul încărcării (Li):

$$Li = 1 - Lb \quad (44)$$

unde, folosind metrica prezentată în [pop], echilibrul încărcării (Lb) este:

$$Lb = \frac{\underset{0 \le i < N_N}{AVG}\ t_{iter}(i, N_B)}{t_{iter}} \quad (45)$$

În această formă, Lb (care ia valori între 0 și 1) este un obiectiv care trebuie maximizat (adus cât mai aproape de 1). Pentru a face o optimizare multicriterială, toate obiectivele trebuie să fie minimizate; de aceea, am ales ca obiectiv Li, și nu Lb.

6.4.3.3.5. Suprafața de referință

Pentru a avea o măsură a eficienței algoritmului, am construit o aproximație a suprafeței Pareto în spațiul celor trei obiective: energie folosită, timp de rulare și încărcare. Vom desemna această suprafață ca Suprafața de referință.

Am dezvoltat un algoritm metaeuristic original pentru a genera o suprafață de referință care aproximează mulțimea Pareto. Algoritmul poate fi descris prin următorul pseudocod:

```
// Calculează valorile pentru timpul de rulare și energia folosită
// de fiecare nod pentru toate configurațiile de frecvență și toate
// dimensiunile de bloc (memorizare)

For i in (0,NN - 1)
   For f_CPU in CPU frequencies
      For f_GPU in GPU frequencies
         For b in dim_blocs
            Calculate ti(f_CPU, f_GPU, b) with formula (32)
            Calculate Ei(f_CPU, f_GPU, b) with formula (33)
            Insert (f_CPU, f_GPU, ti, Ei) in timei dictionary in the sorted list at key b
      Insert (f_CPU, f_GPU, ti, Ei) in Energyi dictionary in the sorted list at key b
         End For b
```

```
        End For f_gpu
    End For f_cpu
End For i
// generează punctele candidat
// generează toate partițiile 16 units pe 16 nodes.
// fiecare partiție este un vector cu 16 poziții
// elementul i conține încărcarea pentru nodul i
scheds←schedules(16,16)
//pornește de la o mulțime referință vidă
ref_set←[]
//pentru fiecare partiție
for s in sched
//se inițializeaza variabilele obiectiv
 tm_s←0
 tm_ms←0
 E_ts←0
 tE_s←0
 tE_ms←0
 E_m←0
// se calculează candidatul cu timpul de rulare minim
        for i in (0,NN – 1)
//găsește în dicționar configurația cu timpul minim pe nodul i
//pentru sarcina s[i]
//lista este sortată ascendent după timp
                (f_tmin_CPU[i], f_tmin_GPU[i], tmin[i],Et[i])=head timei [s[i]]
// găsește în dicționar configurația cu energia minimă pe nodul i
//pentru sarcina s[i];
//lista este sortată ascendent după energie, this is the first element
                (f_Emin_CPU[i], f_Emin_GPU[i], tE[i],Emin[i])=head Energyi [s[i]]
 //actualizează variabilele obiectiv pentru planificarea curentă s
                tm_s←max(tm_s, tmin[i])
                tm_ms←tm_ms+tmin[i])
                E_t←E_t+Et[i]
 tE_s←max(tE_s, tE[i])
                tE_ms←tE_ms+tE[i])
                E_m←E_m+Emin[i]
end for i
        ib_t←1- tm_ms/NN/tm_s
ib_E←1- tE_ms/NN/tE_s

// avem două configurații de inclus în setul de referință ref_set
// configurația cu timpul minim pentru planificarea curentă s
        tmin_nondom←ref_set insert ((s, f_tmin_CPU, f_tmin_GPU)(tm_s,E_t, ib_t))
// configurația cu energia minimă pentru planificarea curentă s
Emin_nondom←ref_set insert ((s, f_Emin_CPU, f_Emin_GPU)(tE_s, E_m, ib_E))

 if tmin_nondom
//dacă candidatul cu tmin este nedominat îl folosim ca punct de plecare
        f_cand_CPUt←f_tmin_CPU
        f_cand_GPUt←f_tmin_GPU
        end if
 if Emin_nondom
// dacă candidatul cu Emin este nedominat îl folosim ca punct de plecare
                f_cand_CPUE←f_Emin_CPU
        f_cand_GPUE←f_Emin_GPU
```

```
        end if

 if tmin_nondom OR Emin_nondom
        for i in (0,NN - 1)
 nondom ←true
 CPU_nondom←true
 GPU_nondom←true
 if tmin_nondom
        while nondom
//pornind de la candidatul tmin, căutăm noi candidați
//prin scăderea frecvenței CPU
        if CPU_nondom
                f_cand_CPU[i]← step-down f_cand_CPU[i]
                calculate tc,Ec,ibc using new f_cand_CPU, f_cand_GPU
        CPU_nondom ← ref_set insert ((s, f_cand_CPU,f_cand_GPU)(tc,Ec, ibc))

 if not CPU_nondom
        f_cand_CPU[i]← step-up f_cand_CPU[i]
                endif
                endif
//prin scăderea frecvenței GPU
 if GPU_nondom
                f_cand_GPU[i]← decrease f_cand_GPU[i]
                calculate tc,Ec,ibc using f_cand_CPU, new f_cand_GPU
        GPU_nondom ← ref_set insert ((s, f_cand_CPU,f_cand_GPU)(tc,Ec, ibc))
        if not GPU_nondom
        f_cand_GPU[i]← step-up f_cand_GPU[i]
                endif
                endif
 nondom← CPU_nondom OR GPU_nondom
 end while
 endif
 nondom ←true
 CPU_nondom←true
 GPU_nondom←true
 if Emin_nondom
        while nondom
//pornind de la candidatul cu Emin, căutăm noi candidați
//prin creșterea frecvenței CPU
        if CPU_nondom
                f_cand_CPU[i]← step-up f_cand_CPU[i]
                calculate tc,Ec,ibc using new f_cand_CPU, f_cand_GPU
        CPU_nondom ← ref_set insert ((s, f_cand_CPU,f_cand_GPU)(tc,Ec, ibc))
 if not CPU_nondom
        f_cand_CPU[i]← step-down f_cand_CPU[i]
                endif
                endif
//prin creșterea frecvenței GPU
 if GPU_nondom
                f_cand_GPU[i]← step-up f_cand_GPU[i]
                calculate tc,Ec,ibc using f_cand_CPU, new f_cand_GPU
        GPU_nondom ← ref_set insert ((s, f_cand_CPU,f_cand_GPU)(tc,Ec, ibc))
        if not GPU_nondom
        f_cand_GPU[i]← step-down f_cand_GPU[i]
                endif
```

endif
nondom← CPU_nondom OR GPU_nondom
end while
endif
end for i
endif
endfor s

6.4.3.3.6. Cazul inițializării

Scopul simulării este:

- să testeze dacă algoritmul dezvoltat poate ajunge la o mapare (descompunerea domeniului plus planificare), care este aproape de setul de referință;
- să determine cât de repede se poate ajunge la această mapare.

Figura 38 prezintă un exemplu de distribuție pentru o descompunere 4x4 a domeniului. În fiecare pas, nodul care anunță este colorat în albastru, nodurile care au terminat distribuția și au început calculele sunt colorate în verde, iar nodurile care nu pot distribui mai departe sunt colorate în roșu. Săgețile prezintă contractele realizate. După ce toate nodurile au început calculul, blocurile sunt, de asemenea, redistribuite în faza de comunicație. Acest lucru se întâmplă până ce timpul de comunicație devine aproximativ egal pe toate nodurile.

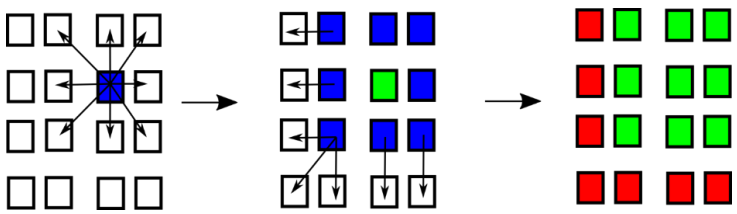


Figura 38. Procesul de distribuire a sarcinii computaționale pe nodurile disponibile [Mironescu2018]

Figura 39 prezintă una din mapările stabile ca pe un punct roșu, comparat cu setul de referință în albastru.

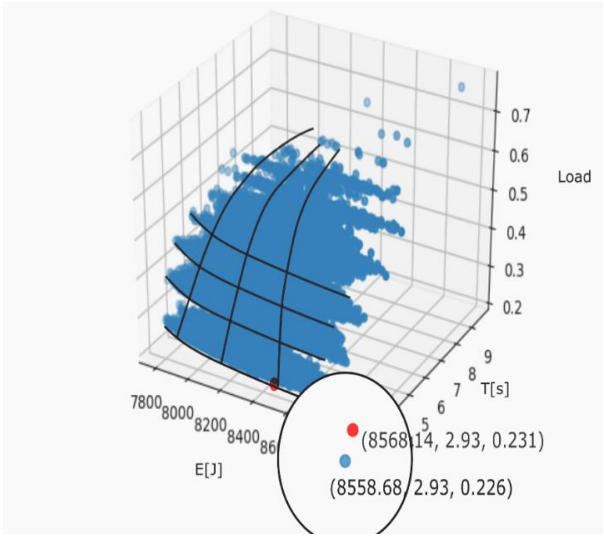


Figura 39. Setul de referință (albastru) și maparea stabilă la care s-a ajuns după redistribuire [Mironescu2018]

Prin analiza jurnalului simulării, s-a descoperit cauza neatingerii unui punct din setul de referință. Redistribuirea se oprește înainte de a atinge setul de referință. Nodul mai rapid a terminat procesarea și a cerut zona de margine. Cererea a ajuns după ce nodul mai încet (lent) transferase ultimele blocuri din cozile de margine în cozile de procesare. În acest fel, ele nu mai pot fi transferate.

6.4.3.3.7. Creșterea sarcinii computaționale

Scopul acestei simulări a fost să arate cum reacționează algoritmul în cazul unei creșteri dinamice a sarcinii computaționale. Această situație apare când metoda numerică impune rafinarea rețelei de discretizare, pentru micșorarea erorilor de aproximare. Rafinarea se face prin înjumătățirea distanțelor dintre punctele de discretizare. În acest fel, se dublează numărul de puncte în ambele direcții. Ca urmare, numărul total de puncte al subdomeniului va fi de patru ori mai mare decât al subdomeniului inițial.

Figura 40 prezintă timpul necesar pentru a ajunge în apropierea soluției optime, iar Figura 41 numărul de iterații necesare pentru a ajunge în apropierea setului de referință pentru un singur nod afectat.

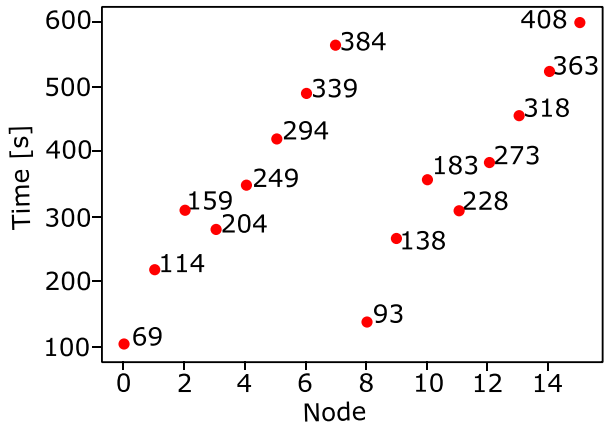


Figura 40. Timpul necesar pentru a ajunge în apropierea soluției optime [Mironescu2018]

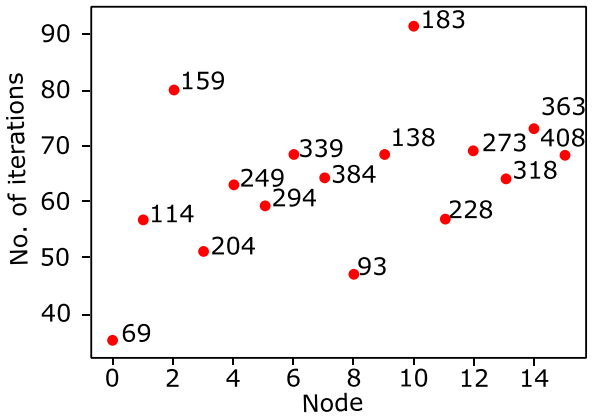
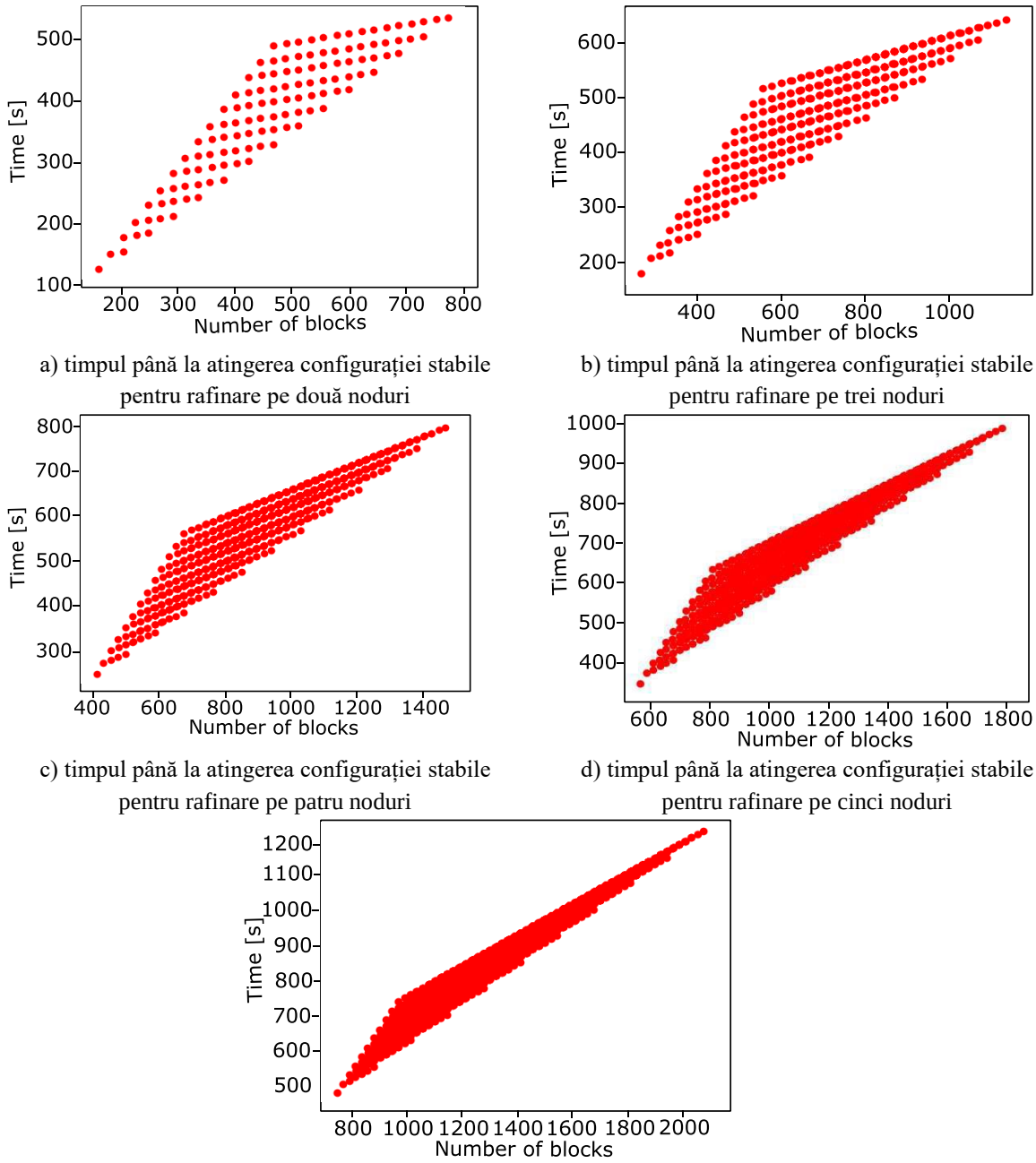


Figura 41. Numărul de iterații necesar pentru a ajunge în apropierea soluției optime [Mironescu2018]

(Eticheta punctului indică numărul de blocuri suplimentare care trebuie redistribuite (3 *încărcarea optimă, conform Tabelului 6))

Încărcarea optimală este NBO(i)=Nr_of_CPU_compute + 8*nr_GPU. Numărul de nuclee CPU care pot fi alocate calculului Nr_of_CPU_compute este dat de numărul total de nuclee CPU disponibile – numărul de CPU-uri necesare pentru rularea driver-ului StarPU pentru GPU.

Figura 42 prezintă timpul necesar pentru a ajunge la o configurație stabilă, în apropierea setului de referință, după ce a fost declanșată rafinarea pe două (42a) până la șase (42e) noduri simultan. Se poate observa că, dependent de combinația de noduri, avem timpi diferiți pentru același număr total de blocuri. Acesta este rezultatul configurației hardware a vecinilor și a numărului lor. Timpul crește odată cu creșterea numărului de noduri, pentru că partajează aceleași canale de comunicație.



a) timpul până la atingerea configurației stabile pentru rafinare pe două noduri

b) timpul până la atingerea configurației stabile pentru rafinare pe trei noduri

c) timpul până la atingerea configurației stabile pentru rafinare pe patru noduri

d) timpul până la atingerea configurației stabile pentru rafinare pe cinci noduri

e) timpul până la atingerea configurației stabile pentru rafinare pe șase noduri

Figura 42. Rafinarea gridului pentru două (a), trei (b), patru (c), cinci (d) și 6 noduri (e) [Mironescu2018]

6.4.3.3.8 Cazul defectării nodurilor

Primul obiectiv urmărit a fost să aflăm dacă sistemul își revine după pierderea unui nod (sau mai multe noduri).

Al doilea obiectiv a fost să vedem în cât timp maparea ajunge în apropierea setului de referință, pentru numărul corespunzător de noduri.

Figura 43 prezintă timpul în care se ajunge la o configurație stabilă, în apropierea setului de referință, pentru 15 noduri, în funcție de numărul de blocuri care trebuie redistribuite.

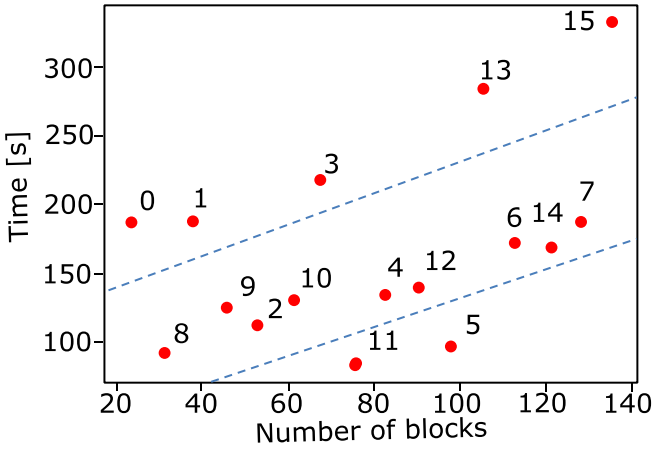


Figura 43. Timpul pentru redistribuirea sarcinii unui nod defect [Mironescu2018]
(Eticheta punctelor reprezintă numărul nodului defect)

Se observă trei clase de comportament al sistemului după defectarea unui nod:

- noduri după care sistemul se recuperează încet: 0, 1, 3, 13, 15;
- noduri după care sistemul se recuperează într-un timp mediu: 8, 9, 2, 10, 4, 12, 6, 14, 7;
- noduri după care sistemul se recuperează rapid: 11, 5.

Rezultatele simulării au furnizat informații cantitative și calitative despre timpul de redistribuire a sarcinii și factorii care îl influențează. Astfel, au fost puse în evidență dependența acestui timp de numărul și configurația vecinilor nodului.

6.5. Modelarea și simularea la scară mare folosind simulatorul SimGrid

Pentru modelare, a fost folosită varianta de Java a mediului pentru dezvoltare de simulatoare SimGrid. Au fost modelate: unitățile de procesare, prin clasele CPUși GPU derivate din clasa Process; componentele aplicației implementate în CHARM++ și StarPU, prin clasa Chare, derivată din Process; rutinele OpenCL, prin clasa Kernel derivată din clasa Task.

Configurația hardware-ului a fost descrisă prin definirea unei colecții de gazde (hosts) cu diverse viteze de procesare, conectate între ele prin legături de comunicație care modelează magistralele interne de comunicație în interiorul nodului și legăturile de rețea între noduri.

Maparea a fost descrisă prin asocierea dintre instanțele claselor și gazde. In interiorul nodului, instanțele clasei Chare (câte una pentru fiecare nod) generează instanțele din clasa Kernel și le distribuie către instanțele claselor CPU și GPU pentru execuție. Prin rețea, instanțele clasei Chare negociază, conform algorimului de mapare cvasi optimală, redistribuirea sarcinii.

O simulare cu 1024 de noduri, cu structura nodurilor Coka, multiplicând de 64 de ori configurația folosită la modelarea cu rețele Petri, a durat 37 de minute pe un sistem cu procesor multicore Pentium I7 cu memorie de capacitate 8 GB DRAM.

6.6. Contribuții originale

A fost definit și dezvoltat un mediu integrat care permite modelarea și simularea sistemului hardware–software, încă din faza de pre-proiectare, la mai multe niveluri de detaliere.

Modelarea și simularea folosind acest mediu are mai multe avantaje față de simulatoarele de arhitecturi clasice din categoria *cycle accurate*:

- **Dimensiunea sistemului modelat**. Pentru investigarea mapării aplicațiilor științifice care exploatează și paralelismul de la nivelul rețelei de inter-conectare, este important să fie modelate și interacțiunile de la acest nivel. În general, simulatoarele de tipul *cycle accurate* sunt limitate la nivelul nucleelor dintr-un singur procesor. Pentru a modela sisteme cu sute de nuclee, aceste simulatoare sunt obligate să recurgă la tehnici de simulare cu evenimente discrete, ceea ce le scade precizia cu până la 25% [sniper]. Modelul prezentat de noi poate aborda sisteme cu 16 noduri heterogene, fiecare cu până la 16 nuclee CPU și 16 procesoare GPU, la un nivel de precizie mai mic, dar suficient pentru aprecierile calitative necesare pentru decizii în faza de pre-proiectare;
- **Timpul necesar pentru simulare**. Pentru ca rezultatele să fie relevante la nivelul de precizie al simulatorului *cycle accurate*, aplicația ar trebui simulată în modul *full-system.* Acest lucru presupune simularea tuturor straturilor software implicate:
  - aplicația;
  - cele două medii de execuție (StarPU, CHARM++);
  - mediile de execuție care asigură firele de execuție (biblioteca POSIX threads pthreads), accesul la dispozitivele acceleratoare(OpenCL) și comunicația prin rețea (Adaptive Message Passing Interface AMPI);
  - sistemul de operare**.**

  Chiar dacă este posibilă cu simulatoare precum simulatorul multicore/ manycore Sniper, rularea în aceste condiții este prohibitiv de înceată pentru a surprinde, de exemplu, o redistribuire completă la nivel de rețea a sarcinii computaționale, ca urmare a rafinării gridului;
- **Flexibilitatea**. De obicei, un simulator din categoria *cycle accurate* implementează doar un limbaj mașină (*Instruction Set Architecture* – ISA) pentru procesoare din aceeași familie. Situațiile în care se implementează și ISA-uri de la dispozitivele eterogene sunt și mai rare (de exemplu, Multi2Sim). Modelul este descris în sursele simulatorului, scrise într-un limbaj de nivel înalt. Chiar și atunci când aceste surse sunt disponibile, implementarea unui nou model, cu un ISA diferit, nu este trivială și poate introduce erori câteodată greu de detectat.

  Modelul propus de noi este permanent accesibil, ceea ce permite modificarea lui chiar și la momentul rulării, iar definirea unor noi modele de unități de procesare este mult mai simplă;
- **Verificare formală a validității.** Modelul dezvoltat în cadrul acestei teze, prezentat în secțiunea 6.4.1 folosește rețele Petri, care pot fi verificate formal, așa că se pot evita erorile și se pot garanta proprietățile sistemului modelat. În secțiunea 6.4.2 au fost prezentate rezultate ale acestui tip de analiză aplicate modelului propus;
- **Capacitatea de a modela și evalua sisteme hardware–software** care nu există decât **sub formă de specificații**. O arhitectură hardware care nu există decât sub formă de specificație poate fi modelată într-un simulator *cycle accurate* dacă specificația este detaliată la nivel de ISA. Software-ul, în schimb, trebuie să existe la nivel de instrucțiuni mașină compilate pentru ISA-ul respectiv (ceea ce, iarăși, nu este întotdeauna trivial; în lumea *embedded,* este dificil de implementat un compilator matur pentru fiecare ISA).

Folosind rețelele Petri, în cadrul acestei cercetări au fost pot modelate, pornind de la specificații date, atât hardware-ul cât și software-ul. Au fost construite modelele pentru mai multe niveluri de detaliere a specificațiilor și s-a putut garanta, prin verificare formală, că modelele respectă specificațiile impuse.

Din toate aceste puncte de vedere, metoda propusă de noi este mult mai adecvată pentru faza de pre-proiectare, comparativ cu abordarea clasică (care utilizează simulatoare de tipul *cycle accurate*).

Dezvoltarea în mediul integrat este facilitată de posibilitatea oferită de mediul CPN Tools de a modela și simula modelul în aceeași fereastră. Lipsa unei faze de compilare a modelului și a unei faze de simulare, asupra căreia nu se poate interveni, are avantaje similare cu cele ale dezvoltării de programe interpretate, și anume:

- Ciclul de dezvoltare este mult mai scurt;
- Se pot face oricând modificări în timpul simulării și se poate evalua direct efectul modificării, pornind de la starea curentă a sistemului (marcajul curent), fără a fi necesară reinițializarea acestuia;
- Se poate aprecia efectul unei modificări în contextul în care acest efect este maxim. Nu trebuie să refacem contextul pentru a sesiza efectul.

În contrast cu metodele care nu au suport formal, rețelele Petri permit atât verificarea corectitudinii modelului, cât și a corectitudinii soluției propuse pentru sistemul modelat. Astfel, în timpul dezvoltării modelului ansamblului hardware–software în cadrul acestei cercetări, au putut fi identificate greșeli:

- în modelul hardware-ului, care generau un comportament fizic imposibil – spre ex., multiplicarea fizică a resurselor;
- în protocolul de comunicare, care ar fi condus și în sistemul modelat la blocaje prompt semnalate în model, prin stări moarte, identificate prin analiză.

Un avantaj suplimentar al metodelor formale și al rețelelor Petri, în special, este că localizarea sursei greșelii este mult mai simplă decât în cazul metodelor clasice. Prin analiza formală, prezentată în secțiunea 6.4.2, s-au identificat marcajul și structura în care are loc blocajul; în felul acesta, s-a determinat modulul hardware sau software care a provocat acest blocaj. În simularea de tip *cycle accurate,* descoperirea cauzelor exacte pentru care rularea se oprește la o anumită linie de cod nu este întotdeauna simplă și nici nu sugerează o posibilitate de soluționare.

Prin folosirea logicilor formale, s-a putut demonstra că sistemul hardware–software modelat parcurge cu siguranță toate etapele corespunzătoare unei iterații din algoritmul numeric implementat în aplicație. Se poate, astfel, garanta formal un anumit tip de comportament pentru sistemul dezvoltat. În cazul simulatoarelor *cycle accurate,* acest lucru nu poate fi garantat pentru sistemele dezvoltate cu ajutorul lor.

Folosind facilitățile de modelare și simulare ale mediului, au fost testate și dezvoltate componentele algoritmului adaptiv (dezvoltat în capitolul 5) pentru maparea dinamică cvasi-optimală a aplicației pe arhitectura HPC generică. În acest context, demersul simulării tuturor planificatoarelor implementate în StarPU cu rețele Petri era singular la momentul publicării rezultatelor în [Mironescu2014], după știința noastră.

Simulările au demonstrat eficiența planificatoarelor din familia *dequeue model* (dm) pentru planificarea task-urilor pe sistemul eterogen, mai ales în condițiile în care acestea sunt

de dimensiuni diferite și au dependențe de date. Rezultatele cele mai bune le are planificatorul *dequeue model data aware ready* (dmdar), care dă prioritate mai mare task-urilor ale căror zone de date au fost deja transferate, astfel că execuția lor maschează transferul celorlalte task-uri. Planificatorul *work stealing (ws)* nu are nevoie de un model de performanță al unităților de execuție și de informație legată de starea lor actuală. El intră în acțiune când dezechilibrul este cert și redistribuie datele acolo unde există resurse pentru procesarea lor, ceea ce asigură întotdeauna o îmbunătățire a echilibrării. Ca urmare, *ws* este un planificator robust și eficient, dacă se face raportul efort suplimentar/ performanță.

Aceste concluzii au fost folosite pentru dezvoltarea propriului algoritm de mapare cvasi-optimală. Algoritmul rezultat a fost investigat prin simulare folosind modelul prezentat în secțiunea 6.4.1 si datele obținute prin simularea cu simulatorul Multi2Sim. Rezultatele simulării arată că planificatorul poate scurta timpul de calcul cu 15%.

Algoritmul de mapare cvasi-optimală a fost extins pentru optimizarea multicriterială. Pentru testarea algorimului prin simulare, au fost folosite date din literatură [Calore2017] [Martinez2009] [van Werkhoven2014] pentru consumul de timp și energie la rularea unei aplicații bazate pe același algoritm numeric *(Lattice Boltzmann*) pe un sistem de tip High Performance Computing heterogen [coka].

Pentru evaluarea rezultatelor simulării, am dezvoltat un model matematic original care permite estimarea timpului și a consumului energetic la calculul unei iterații din metoda numerică *Lattice Boltzmann* pentru o configurație dată a unui nod eterogen și o dimensiune de bloc dată.Prin folosirea acestui model pentru toate nodurile unui cluster eterogen cu o anumită configurație și pentru o anumită distribuție a unui domeniu, pe noduri se pot calcula valorile obiectivelor de optimizare (timpul de calcul, energia consumată și dezechilibrul încărcării computaționale, conform secțiunii 6.4.3.3.4, formulele 41, 42, 44) pentru distribuția dată. Aceste valori obiectiv pot fi folosite în alți algoritmi, de exemplu ca funcție de fitness a algoritmilor genetici de optimizare multicriterială, ceea ce ar permite adoptarea în viitor a acestor metode.

Modelul matematic a fost folosit la dezvoltarea unui algoritm original pentru găsirea unei aproximări a suprafeței Pareto corespunzătoare problemei de optimizare multiobiectiv a mapării aplicației pe o arhitectură hardware data, de tip HPC. În comparație cu metodele genetice de optimizare multicriterială, algoritmul are următoarele avantaje:

- are o implementare mai simplă și mai ușor de integrat într-o platformă de echilibrare a încărcării;
- nu are nevoie de expertiză pentru codificarea candidaților și a operațiilor definite pe aceștia (mutații, crosover). Algoritmul nu are nevoie decât de funcția care generează performanța configurației, de care, de altfel, are nevoie și algoritmul genetic, ca funcție de fitness;
- este capabil să selecteze măcar anumiți indivizi din suprafața Pareto;
- poate folosi gradientul în căutările locale, ceea ce metodele genetice simple nu pot.

A fost testată prin simulări capacitatea de redistribuire dinamică a algoritmului de mapare cvasi-optimală în situații normale (iterații curente) sau speciale (inițializare, creșterea sarcinii datorită rafinării discretizării, creșterea sarcinii la căderea unui nod). Simulările au demonstrat că algoritmul de mapare are capacitatea de a trata toate aceste situații și generează o mapare în apropierea aproximării setului Pareto.

Față de soluția implementată în aplicația walLBerla [Feichtinger2014], care are un caracter static, algoritmul propus în această teză de doctorat pentru redistribuirea sarcinii la rafinarea rețelei de discretizare are un caracter dinamic adaptiv, astfel încât investigarea lui prin modelare și simulare reprezintă o contribuție originală.

Rezultatele simulării au furnizat informații cantitative și calitative despre timpul de redistribuire a sarcinii și factorii care îl influențează. Astfel, au fost puse în evidență dependența acestui timp de numărul și configurația vecinilor pe care îi are nodul.

Implementările curente ale algoritmilor de planificare nu includ și mecanisme de recuperare în caz de defectare, pentru că nu se bazează pe un protocol de comunicare și negociere care să permită implementarea acestor mecanisme. Acest lucru se întâmplă și pentru că se încearcă rezolvarea problemelor prin mecanisme mai generale, de exemplu prin virtualizare. Ca urmare, abordarea noastră este originală, rezolvă problema mai rapid și prezintă avantajul că nu necesită mecanisme suplimentare exterioare aplicației pentru recuperare și nici întreruperea calculelor.

7. CONCLUZII, CONTRIBUȚII ORIGINALE ȘI PERSPECTIVE

În această cercetare, am urmărit dezvoltarea unui sistem integrat pentru maparea cvasi-optimală a unor aplicații științifice pe arhitecturi HPC (High Performance Computing) parametrizabile.

Ținând cont de obiectivele propuse, în cadrul tezei au fost realizate:

✓ A fost dezvoltată o platformă, care are instrumente specifice MDE (*Model Driven Engineering)*:
- Modele descriptive în limbajul SysML, la nivelul aplicației, hardware-ului și mapării;
- Transformări între modele (M2M);
- Transformări de la model la text (M2T)

ca elemente integratoare.

Utilizarea acestor instrumente a asigurat:
- Gestionarea eficientă a procesului de generare a tuturor fișierelor necesare pentru simularea variantelor constructive;
- Folosirea integrată a unei ierarhii de simulatoare.

Platforma este prezentată în capitolul 3 al lucrării.

Principala contribuție originală adusă de această abordare este aplicarea metodelor de tip MDE, prin intermediul instrumentelor MDE, la maparea optimă a unei aplicații pe un sistem de calcul HPC parametrizabil. Actual, aceste metode sunt folosite la dezvoltarea hardware–software a sistemele încorporate (embedded).

✓ A fost dezvoltată o arhitectură originală de aplicație software, care combină două medii de execuție (StarPU și CHARM++) pentru programe concurente.

Fiecare dintre aceste medii este conceput pentru exploatarea eficientă a doar unuia dintre cele două niveluri de organizare ale unui sistem HPC. Astfel, CHARM++ facilitează distribuirea aplicației între nodurile care comunică prin rețea, iar StarPU facilitează distribuirea aplicației pe unitățile de procesare eterogene ale unui nod. Față de o aplicație dezvoltată folosind doar CHARM++, aplicația cu arhitectura propusă de autorul tezei are avantajul

distribuirii mai eficiente pe sisteme eterogene, pe care *chare*-urile (obiectele alocabile din CHARM++) nu se pot mapa ușor.

Arhitectura sofware (prezentată în capitolul 4) se poate adapta la particularitățile diferitelor niveluri de organizare ale arhitecturii hardware, astfel încât să mapeze cât mai bine concurența exprimată în software, pe paralelismul disponibil în hardware. Astfel, comparativ cu o aplicație care folosește doar StarPU, arhitectura propusă în cadrul acestei teze permite implementarea unor mecanisme mai sofisticate de comunicație între noduri, bazate pe modelul Actor, suportat de CHARM++. Acest avantaj a fost folosit pentru implementarea unui sistem de echilibrare a încărcării între noduri, bazat pe un protocol de negociere (prezentat în capitolul 5).

În cadrul arhitecturii originale de aplicație software, componenta MDE poate gestiona complexitatea dată de numărul mare de fișiere, în comparație cu aplicațiile dezvoltate folosind limbajele care folosesc directive pentru specificarea paralelismului (la care rularea și testarea sunt mai complicate).

Folosirea acestei arhitecturi a permis separarea codului aplicației în:

- Rutine OpenCL, în care este implementat algoritmul numeric. Rutinele OpenCL au fost editate independent și apoi rulate pe arhitectura reală și pe simulatoare de tip *cycle accurate* (Multi2Sim). Rutinele au fost optimizate pe baza rezultatelor rulărilor. Codul optimizat al rutinelor a fost asociat apoi cu modelul descriptiv din platforma MDE;
- Cod StarPU, în care este implementat planificatorul pentru distribuirea task-urilor pe nod. La nivelul StarPU, au fost dezvoltate și testate planificatoarele care distribuie sarcina computațională pe nodul eterogen. Astfel, au fost investigate prin simulare performanțele planificatoarelor implementate în StarPU în distribuirea sarcinii computaționale a aplicației (proces prezentat în capitolul 6). Rezultatele simulării au fost folosite pentru dezvoltarea propriului planificator pentru nivelul nodului eterogen (prezentat în capitolul 5);
- Cod CHARM++, în care este implementată comunicația între noduri. La nivel CHARM++, au fost dezvoltate:
    - mecanismul care asigură comunicarea necesară algoritmului numeric;
    - protocolul de comunicare care implementează planificatorul dinamic difuziv de la nivelul rețelei de noduri (prezentat în capitolul 5).

✓ A fost dezvoltat un model holistic original al ansamblului hardware–software.

Modelarea arhitecturii hardware–software a fost realizată la un nivel de detaliu care permite evaluarea gradului în care aplicația exploatează paralelismul hardware-ului, atât la nivelul nodului, cât și la nivelul rețelei de interconectare a nodurilor.

Folosirea acestui model (prezentat în capitolul 6) a permis:

- Simularea unui sistem cu 16 noduri heterogene, fiecare cu până la 16 nuclee CPU și 16 procesoare GPU. Nivelul de precizie al modelului dezvoltat în cadrul tezei este mai mic decât la simulatoarele de tipul *cycle accurate*, dar suficient pentru aprecierile calitative necesare pentru decizii în faza de pre-proiectare;
- Simularea simultană a aplicației și a celor două medii de execuție (StarPU, CHARM++). De asemenea, au fost incluse, într-o formă abstractizată, și celelalte

straturi sofware implicate. Acest lucru este mult mai dificil de realizat într-un simulator *cycle accurate,* care nu permite abstractizarea anumitor straturi;

- Dezvoltarea flexibilă, datorată:
  - disponibilității permanente a modelului, care permite modificarea lui chiar și la momentul rulării;
  - ușurinței în definirea unor noi modele de unități de procesare, comparativ cu simulatoarele din categoria *cycle accurate* (care implementează doar un limbaj mașină pentru procesoare din aceeași familie);
  - ciclului scurt de dezvoltare;
- Verificarea formală a validității - acest model a fost verificat formal, ceea ce a permis evitarea erorilor și garantarea finalizării execuției aplicației;
- Modelarea și evaluarea sistemului hardware–software, aflat încă sub formă de specificații.

✓ A fost verificată și evaluată arhitectura hardware–software dezvoltată în această lucrare. Prin verificarea și evaluarea arhitecturii hardware–software (capitolul 6), au putut fi identificate următoarele aspecte:

- Localizarea sursei greșelii a fost mai ușoară, comparativ cu metodele clasice; la analiza formală, s-au identificat exact marcajul și structura care au determinat blocajul; în felul acesta, s-a determinat modulul hardware sau software care a provocat acest blocaj. Prin verificarea formală a modelului, au putut fi identificate greșeli:
  - în modelul hardware. Greșelile generau un comportament fizic imposibil – multiplicarea fizică a resurselor;
  - în protocolul de comunicație. Greșelile ar fi condus și în realitate la blocaje prompt semnalate în model prin stări moarte identificate prin analiză;
- S-a demonstrat că sistemul modelat ajunge cu siguranță într-o anumită stare. Acest lucru nu poate fi garantat formal de simulatoarele *cycle accurate* pentru sistemele modelate și optimizate cu ajutorul lor;
- A fost investigată prin simulare eficiența algoritmilor implementați în mediul StarPU:
  - S-a evidențiat eficiența algoritmilor din familia *dequeue model* (*dm*) pentru planificarea pe sistemul eterogen modelat a task-urilor de dimensiuni diferite și cu dependențe de date. În acest context, algoritmul *dequeue model data aware ready (dmdar)* a avut cele mai bune rezultate, deoarece a dat prioritate mai mare task-urilor ale căror zone de date au fost deja transferate, astfel că execuția lor a mascat transferul celorlalte task-uri;
  - S-a evidențiat că algoritmul *work stealing* (*ws*) este robust și eficient din punct de vedere al raportului efort suplimentar/ performanță, deoarece nu are nevoie de un model de performanță al unităților de execuție și de informație legată de starea lor actuală. El intră în acțiune când dezechilibrul este cert și redistribuie datele acolo unde există resurse libere pentru procesarea lor, ceea ce asigură întotdeauna o îmbunătățire a echilibrării.

În urma verificării și evaluării arhitecturii hardware–software, cei doi algoritmi (unul de tipul *dequeue model* și unul de tipul *work stealing*) au fost combinați, dezvoltându-se propriul algoritm la nivelul nodului (prezentat în Capitolul 5). La nivelul rețelei, a fost ales un algoritm de tipul *work stealing*. Algoritmul de mapare original rezultat prin cuplarea celor doi algoritmi (de la nivelul nodului și de la nivelul rețelei) a fost investigat prin simulare; rezultatele simulării au arătat că noul algoritm (algoritm de mapare cvasi-optimală) poate scurta timpul de calcul cu 15%.

Rezultatele simulării au furnizat informații cantitative și calitative despre timpul de redistribuire a sarcinii și factorii care îl influențează. Astfel, a fost pusă în evidență dependența acestui timp de numărul și configurația vecinilor pe care îi are nodul.

Față de soluția implementată în aplicația walLBerla [Feichtinger2014], care are un caracter static, algoritmul de mapare propus de noi pentru redistribuirea sarcinii la rafinarea rețelei de discretizare are un caracter dinamic adaptiv, așa încât investigarea lui prin modelare și simulare reprezintă o contribuție originală.

Considerăm o contribuție nouă, studiul prin simulare a toleranței sistemului la defectare. Studiul nu a mai fost efectuat la acest nivel de detaliu și în contextul unui algoritm de mapare. Abordările curente de fiabilitate folosesc modele simplificate ale hardware-ului si software-ului și se referă la mecanisme mai generale de recuperare.

- ✓ A fost realizată optimizarea multi-obiectiv a mapării hardware–software folosind algoritmul de mapare cvasi-optimală.

  Algoritmul de mapare cvasi-optimală (prezentat în Capitolul 5) a fost obținut prin combinarea altor doi algoritmi:

  - Un algoritm de distribuție dinamică a sarcinii pe unitățile de procesare eterogene ale fiecărui nod. Acest algoritm combină, la rândul lui, două tipuri de planificare:
    - o planificare predictivă, care folosește un algoritm din familia *dequeue model* (*dm*) pentru a estima pe care unitate să plaseze sarcina;
    - o planificare reactivă, care sesizează un dezechilibru în încărcarea unităților de procesare și încearcă să redistribuie sarcina printr-un algoritm de *work stealing* (*ws*).

      Combinația între cele două tipuri de planificare este originală și elimină din potențialele neajunsuri ale fiecărei planificări, în parte.
  - Un algoritm difuziv de distribuție dinamică a sarcinii computaționale între nodurile unui sistem de calcul HPC de tip cluster. Acest algoritm a fost implementat ca un protocol de negociere, ceea ce este un aspect original pentru un algoritm difuziv.

  Cuplarea originală a celor doi algoritmi, de la nivelul nodului și de la nivelul rețelei, a permis transferul la alte noduri din rețea a sarcinii computațională pe care nici una din unitățile de procesare locale nu o poate prelua.

  Algoritmul de mapare cvasi-optimală obținut a fost suprapus peste comunicația normală de la sfârșitul unei iterații, cu avantajele: detectarea dezechilibrului în încărcare, indiferent de cauză; nu generează comunicație suplimentară; nu folosește resurse suplimentare pentru a face calcule pentru destinația și dimensiunea transferului pe baza unor date incerte.

  Evaluarea performanțelor algoritmului de mapare cvasi-optimală a fost realizată astfel:

- A fost dezvoltat un model matematic original, care permite estimare timpului și a consumului energetic la calculul unei iterații din metoda numerică Lattice Boltzmann, pentru o configurație dată a unui nod eterogen și o dimensiune de bloc dată.

  Prin folosirea acestui model (descris în capitolul 6), s-au calculat valorile obiectivelor de optimizare (timp de calcul, energie și dezechilibru) pentru distribuția dată. Aceste valori ale obiectivelor de optimizare pot fi folosite în alți algoritmi, de exemplu ca funcție de fitness a algoritmilor genetici de optimizare multicriterială, ceea ce permite adoptarea în viitor și a acestor metode pentru găsirea variantelor optime de mapare.
- A fost dezvoltat un algoritm original pentru găsirea unei aproximări a suprafeței Pareto corespunzătoare problemei multicriteriale de optimizare a mapării. În comparație cu metodele genetice de optimizare multicriterială, algoritmul are avantajele:
  - are o implementare mai simplă și mai ușor de integrat într-o platformă de echilibrare;
  - nu are nevoie de expertiză pentru codificarea candidaților și a operațiilor pe ei (mutații, crossover). Algoritmul nu are nevoie decât de funcția care generează performanța configurației de care are nevoie și algoritmul genetic, ca funcție de fitness;
  - este capabil să selecteze cel puțin câțiva indivizi din suprafața Pareto;
  - poate folosi gradientul în căutările locale, ceea ce metodele genetice simple nu pot.

Prin aceste contribuții, considerăm că a fost atins scopul acestei cercetări, acela de a dezvolta un sistem integrat pentru maparea cvasi-optimală a unor aplicații științifice pe arhitecturi HPC parametrizabile.

Câteva direcții posibile de continuare a cercetărilor ar putea fi următoarele:

- Simulări cu modelul SimGrid la nivel de Supercalculator cu 10.000-100.000 noduri, pentru studierea scalabilității algoritmilor de planificare propuși;
- Obținerea unei implementări a aplicației la scară HPC (10.000 noduri), folosind facilitățile oferite de serviciul de calcul în cloud Amazon Elastic Compute EC2, care să poată fi generat cu instrumentele *Model Driven Engineering* și verificarea modelelor din punct de vedere al acurateții;
- Dezvoltarea unui simulator ierarhic care să cupleze cele trei simulatoare folosite în teză (Multi2Sim, CPNTools, SimGrid), astfel încât să fie posibile simulări simultane la cele trei niveluri (unități de procesare, nod heterogen, rețea de interconectare);
- Analiza formală sistematică a corectitudinii modelelor implementate, folosind instrumentele puse la dispoziție de rețelele Petri și simulatorul SimGrid;
- Investigarea altor instrumente de simulare pentru modelele cu rețele Petri, folosind același formalism, care să permită extinderea nivelului de detaliere și a dimensiunii modelelor. Există studii referitoare la transformarea rețelelor Petri în cod SystemC [Rust2003], precum și cercetări legate de dezvoltarea de simulatoare tip *cycle accurate* bazate pe SystemC [Kohl2016].

BIBLIOGRAFIE

1. [Abts2011] Abts D., Kim J. (2011). High Performance Datacenter Networks: Architectures, Algorithms, and Opportunities. Morgan & Claypool Publishers,2011 10.2200/S00341ED1V01Y201103CAC014.
2. [Acun2016] Acun B., Langer A., Meneses E., Menon H., Sarood O., Totoni E., and Power K. L. (2016). Reliability, and Performance: One System to Rule them All. IEEE Computer, 49 (10): 30-37.
3. [Agha1986] Agha G. (1986). Actors: A Model of Concurrent Computation in Distributed Systems. Doctoral Dissertation. MIT Press.
4. [Ahmad2016] Ahmad, W., Yildiz, B.M., Rensink, A., Stoelinga, M. (2016). A Model-Driven Framework for Hardware-Software Co-design of Dataflow Applications. CyPhy 2016: Cyber Physical Systems. Design, Modeling, and Evaluation: 1-16.
5. [Alankrutha2014] Alankrutha P., Deepika H. V., Natampalli M., Chandra Babu S. N. (2014). Multi-accelerator cluster runtime adaptation for enabling discrete concurrent-task applications. Souvenir of the 2014 IEEE International Advance Computing Conference, IACC 2014: 754-760. 10.1109/IAdCC.2014.6779418.
6. [Alexandrov1995] Alexandrov A., Ionescu M. F., Schauser K. E., Scheiman C. (1995). LogGP: incorporating long messages into the LogP model—one step closer towards a realistic model for parallel computation. In *Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures* (SPAA '95). ACM, New York, NY, USA: 95-105.
7. [Arabnejad2014] Arabnejad H., Barbosa J. G. (2014). List scheduling algorithm for heterogeneous systems by an optimistic cost table. IEEE Transactions on Parallel and Distributed Systems, 25 (3): 682–694,
8. [Augonnet2011] Augonnet C., Thibault S., Namyst R., Wacrenier P.-A. (2011). StarPU: a unified platform for task scheduling on heterogeneous multicore architectures. Concurrency and Computation: Practice & Experience, 23 (2): 187-198.
9. [Bak2017] Bak S., Menon H., White S., Diener M., Kale L. (2017). Integrating OpenMP into the Charm++ Programming Model. ESPM2, Denver, Colorado, USA: 7p. 10.1145/3152041.3152085, http://charm.cs.illinois.edu/newPapers/17-08/a4-bak.pdf
10. [Bak2018] Bak S., Menon H., White S., Diener M., Kale L. (2018). Multi-Level Load Balancing with an Integrated Runtime Approach. 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID 2018, Washington, DC, USA, May 1-4: 31-40.
11. [Bezanson2017] Bezanson, J., Edelman A., Karpinski S., Shah Julia V. B. (2017). A Fresh Approach to Numerical Computing. SIAM Review, 59: 65–98. doi: 10.1137/141000671. url: http://julialang.org/publications/julia-fresh-approach-BEKS.pdf.
12. [Blagojevic2008] Blagojevic F., Feng X., Cameron K.W., Nikolopoulos D.S. (2008). Modeling Multi-Grain Parallelism on Heterogeneous Multi-core Processors: A Case Study of the Cell BE. HiPEAC'08 Proceedings of the 3rd international conference on High performance embedded architectures and compilers, Springer-Verlag Berlin, Heidelberg, 38-52.
13. [Blätke2015] Blätke M.A., Heiner M., Marwan W. (2015). Engineering with Petri Nets. In: Algebraic and Discrete Mathematical Methods for Modern Biology, (R Robeva, Ed.), Elsevier Inc.: 141–193.
14. [Bonachea2017] Bonachea D., Hargrove P. (2017). GASNet Specification, v1.8.1. In: GASNet Specification, v1.8.1. Lawrence Berkeley National Laboratory. LBNL Report #: LBNL-2001064. Retrieved from https://escholarship.org/uc/item/03b5g0q4
15. [Brahambhatt2015] Brahambhatt M., Panchal D. (2015). Comparative Analysis on Heuristic Based Load Balancing Algorithms in Grid Environment. International Journal of Engineering Research & Technology (IJERT), 4 (04): 802-806
16. [Broquedis2010] Broquedis F, Clet-Ortega J., Moreaud S., Furmento N., Goglin B., Mercier G., Thibault S., Namyst R.. hwloc: a Generic Framework for Managing Hardware Affinities in HPC Applications. In Proceedings of the 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP2010), Pisa, Italia, February 2010. IEEE Computer Society Press. https://hal.inria.fr/inria-00429889
17. [Brucker1998] Brucker P. (1998). Scheduling algorithms. 2nd edition, Springer, Heidelberg
18. [Bruel2018] Bruel J.-M., Gerard S. (2018). SysML in Action with Papyrus, preprint

19. [Benveniste2010] Benveniste A., Caillaud B., Pouzet M. (2010). The fundamentals of hybrid systems modelers. 49th IEEE Conference on Decision and Control (CDC), Atlanta, GA: 4180-4185. doi: 10.1109/CDC.2010.5717614
20. [Calborean2013] Calborean H., Jahr R., Ungerer T., Vintan L. (2013). A Comparison of Multi-Objective Algorithms for the Automatic Design Space Exploration of a Superscalar System. Advances in Intelligent Control Systems and Computer Science, Springer Berlin Heidelberg, 187: 489-502.
21. [Calore2017] Enrico C., Alessandro G., Sebastiano F. S., Raffaele T. (2017). Evaluation of DVFS techniques on modern HPC processors and accelerators for energy-aware applications. Concurrency and Computation: Practice and Experience, 29 (12): e4143.
22. [Casanova2014] Casanova H., Giersch A., Legrand A., Quinson M., Suter F. (2014). Versatile, Scalable, and Accurate Simulation of Distributed Applications and Platforms. Journal of Parallel and Distributed Computing, Elsevier, 74: 2899-2917.
23. [Chamberlain2015], Chamberlain B. L. (2015). Chapel. In: Programming Models for Parallel Computing, Balaji P. (ed), MIT Press:129-159.
24. [Chekuri1998]Chekuri, C., APPROXIMATION ALGORITHMS FOR SCHEDULING PROBLEMS, Phd thesis,Stanford university,1998
25. [Chandrasekaran2017] Chandrasekaran S., Beyer J., Juckeland G. (2017). Open ACC in a nutshell. In: Chandrasekaran S. And Juckeland G. (eds.), OpenACC for Programmers Concepts and Strategies, Addison Wesley.
26. [Chen1998] Chen B., Potts C.N., Woeginger G.J. (1998). A Review of Machine Scheduling: Complexity, Algorithms and Approximability. In: Du DZ., Pardalos P.M. (eds), Handbook of Combinatorial Optimization. Springer, Boston, MA.
27. [Chitra2011] Chitra P., Rajaram R., Venkatesh P. (2011). Load balanced reliable task scheduling algorithm for heterogeneous systems. Journal of High Speed Networks, 18 (1): 33-45.
28. [Contreras2008] Contreras G., Martonosi M. (2008). Characterizing and improving the performance of Intel Threading Building Blocks . IEEE Int'l Symp. on Workload Characterization. IISWC), 10 p. http://parsec.cs.princeton.edu/publications/contreras08tbb.pdf
29. [Cook2017] Cook P. (2017) Introduction to Parallel Programming with OpenMP, PThreads and MPI. Cook's Books
30. [Daily2014] Daily J., Vishnu A., Palmer B., van Dam H., Kerbyson D. (2014). On the use of MPI as a PGAS Runtime. International Conference on High Performance Computing (HiPC).
31. [Danskin2016] Danskin J., Foley D. (2016). Pascal GPU with NVLink. 2016 IEEE Hot Chips 28 Symposium (HCS), Cupertino, CA, 2016, 1-24,doi: 10.1109/HOTCHIPS.2016.7936202.
32. [DaSilva2015] da Silva A. R. (2015). Model-driven engineering: A survey supported by the unified conceptual model, Computer Languages, Systems & Structures, 43: 139-155. https://doi.org/10.1016/j.cl.2015.06.001.
33. [David2015] David A., Larsen K.G., Legay A., Mikučionis M., Poulsen D.B. (2015). Uppaal SMC Tutorial, International Journal on Software. Tools for Technology Transfer (STTT), 17(4): 397-415.
34. [DHollander2013] D'Hollander, E., Dongarra, J., Foster, I., Grandinetti, L., & Joubert, G. (Eds.). (2013). Transition of HPC towards exascale computing. Advances in Parallel Computing. Amsterdam, The Netherlands: IOS Press.
35. [Dongarra2003] Dongarra J.J., Luszczek P., Petitet A. (2003). The LINPACK Benchmark: past, present and future. Concurrency and Computation: Practice and Experience, John Wiley & Sons, Ltd.: 803–820
36. [Dongarra2005] Dongarra J., Sterling T., Simon H., Strohmaier E. (2005). High Performance Computing: Clusters, Constellations, MPPs, and Future Directions. Computing in Science and Engineering, 7 (2): 16 p. http://www.ic.unicamp.br/~cortes/mo601/artigos_referencias/dongarra_03_clusters_mpp_constellations.pdf
37. [Ebaid2010] Ebaid A., Ammar R.A., Rajasekaran S., Fergany T.A. (2010). Task clustering & scheduling with duplication using recursive critical path approach (RCPA). The 10th IEEE International Symposium on Signal Processing and Information Technology: 34-41.
38. [Eck2017] Eck C., Garcke H., Knabner P. (2017). Mathematical Modeling, Springer Undergraduate Mathematics Series. Springer International.

39. [Esbai2015] Esbai R., Erramdani M. (2015). Model-to-model transformation in approach by modeling: From UML model to Model-View-Presenter and Dependency Injection patterns, *2015 5th World Congress on Information and Communication Technologies (WICT)*, Marrakech, 1-6, doi: 10.1109/WICT.2015.7489648.
40. [Feichtinger2014] Feichtinger C., Habich, J., Köstler H., Rüde U., Aoki T. (2014). Performance Modeling and Analysis of Heterogeneous Lattice Boltzmann Simulations on CPU-GPU Clusters. Parallel Computing, 46: 1-13.
41. [Fenandez2017] Fenández-Tena A., Marcos C.A., Martínez C., Walters D. K. (2017). A new adaptive time step method for unsteady flow simulations in a human lung, Computer Methods in Biomechanics and Biomedical Engineering, 20:8, 915-917, DOI: 10.1080/10255842.2017.1314469
42. [Fernandez2014] Fernández A., Beltran V., Martorell X., Badia R. M., Ayguadé E., Labarta J. (2014). Task-Based Programming with OmpSs and Its Application. Euro-Par 2014: Parallel Processing Workshops LNCS 8806: 601-612. 10.1007/978-3-319-14313-2_51.
43. [Fietz2012] Fietz J., Krause M.J., Schulz C., Sanders P., Heuveline V. (2012). Optimized Hybrid Parallel Lattice Boltzmann Fluid Flow Simulations on Complex Geometries. In: Euro-Par 2012 Parallel Processing, Kaklamanis C., Papatheodorou T. and Spirakis P.G. (Eds.). Vol. 7484. Lecture Notes in Computer Science. Springer Berlin Heidelberg: 818–829.
44. [Filgueras 2014] Filgueras A., Gil E., Jimenez-Gonzales D., Alvarez C., Martorell X., Langer J., Noguera J., Viseers K. (2014). OmpSs@Zynq All-programmable SoC Ecosystem. Proceedings of ACM/SIGDA Int. Symp. on Field-programmable Gate Arrays, FPGA '14, NY, USA: 137-146.
45. [Florea2014] Florea A., Buduleci C., Chiş R., Gellert A., Vinţan L. (2014). Enhancing the Sniper simulator with thermal measurement, 2014 18th International Conference on System Theory, Control and Computing (ICSTCC), Sinaia: 31-36.doi: 10.1109/ICSTCC.2014.6982386.
46. [Fortuin2016] Fortuin S. (2016). Model Driven Engineering: the decomposition of environments, Master Thesis, Utrecht University.
47. [Frenzel2016] Frenzel L. E. (2016). Chapter 45 - InfiniBand (IB). In Handbook of Serial Communications Interfaces, Newnes, Frenzel L. E.(ed). 181-183, https://doi.org/10.1016/B978-0-12-800629-0.00045-0.
48. [Friese2012] Friese R., Brinks T., Oliver C., Siegel H., Maciejewski A. (2012). Analyzing the trade-offs between minimizing makespan and minimizing energy consumption in a heterogeneous resource allocation problem. In: INFOCOMP 2012, The Second International Conference on Advanced Communications and Computation: 81–89.
49. [Garcia2017] Garcia-Gasulla M. (2017). DLB - Dynamic Load Balancing for Hybrid Applications. PhD Thesis, Barcelona
50. [Gaster2011] Gaster B., Howes L., Kaeli D., Mistry P., Schaa D. (2011). Heterogeneous Computing with OpenCL, Morgan Kaufmann Publishers Inc. San Francisco.
51. [Gautier2013] Gautier T., Lima J. V. F., Maillard N., Raffin B. (2013). XKaapi: A Runtime System for Data-Flow Task Programming on Heterogeneous Architectures. In: Proc. of the 27-th IEEE International Parallel and Distributed Processing Symposium (IPDPS), Boston, USA.
52. [Georganas2016] Georganas E. (2016). Scalable Parallel Algorithms for Genome Analysis. PhD thesis, EECS Department, University of California, Berkeley.
53. [Gershenfeld2011] Gershenfeld N. (2011). The Nature of Mathematical Modeling. Massachusetts Institute of Technology
54. [Grant2015] Grant R., Rashti M., Balaji P., Afsahi A. (2015). Scalable Network Communication Using Unreliable RDMA. In: Handbook on Data Centers, Khan S., Zomaya A. (eds). Springer, New York, NY
55. [Götz2012] Götz J., Donath S., Feichtinger C., Iglberger K., Köstler H., Rüde U. (2012). waLBerla: Simulation of Complex Flows on Supercomputers. In: NIC Symposium 2012 - Proceedings, FZ Jülich, S. 349-356 (NIC Series).
56. [Gropp2014] Gropp W., Hoefler T., Thakur R., Lusk E. (2014). Using Advanced MPI: Modern Features of the Message-Passing Interface. The MIT Press.
57. [Hahnfeld2018] Hahnfeld J., Terboven C., Price J., Pflug H.J., Müller M.S. (2018). Evaluation of Asynchronous Offloading Capabilities of Accelerator Programming Models for Multiple Devices. In:) Accelerator Programming Using Directives, Chandrasekaran S., Juckeland G. (eds). WACCPD 2017. Lecture Notes in Computer Science, vol 10732. Springer.

58. [Hassan2013] Hassan N.M.S., Khan M.M.K., Rasul M.G. (2013). Temperature Monitoring and CFD Analysis of Data Centre. Procedia Engineering, 56: 551-559. https://doi.org/10.1016/j.proeng.2013.03.159.
59. [Heiner2012] Heiner M., Herajy M., Liu F., Rohr C., Schwarick M. (2012). Snoopy – a unifying Petri net tool. In: Proc. PETRI NETS 2012, Hamburg, Springer, LNCS, 7347: 398–407.
60. [Heiner2013] Heiner M., Rohr C., Schwarick M. (2013). MARCIE - Model checking And Reachability analysis done effiCIEntly. In: Proc. PETRI NETS 2013, Milano, Springer, LNCS, 7927: 389–399.
61. [Heiner2015] Heiner M., Schwarick M., Wegener J. (2015). Charlie – an extensible Petri net analysis tool. In: Proc. PETRI NETS 2015, Brussels, Springer, LNCS, 9115: 200–211.
62. [Heinz2011] Heinz S. (2011). Mathematical Modeling, Springer.
63. [Herjay2014] Herajy M., Heiner M. (2014). A steering server for collaborative simulation of quantitative Petri nets. In: Proc. PETRI NETS 2014, Tunis, Springer, LNCS, 8489: 374–384.
64. [Herajy2017] Herajy M., Liu F., Rohr C., Heiner M. (2017). Snoopy's hybrid simulator: a tool to construct and simulate hybrid biological models, BMC Systems, Biology.
65. [Hoefler2009] Hoefler T., Schneider T., Lumsdaine A. (2009). LogGP in Theory and Practice - An In-depth Analysis of Modern Interconnection Networks and Benchmarking Methods for Collective Operations. Elsevier Journal of Simulation Modelling Practice and Theory (SIMPAT). 17 (9): 1511-1521.
66. [Hong2010] Hong S., Kim H. (2010). An integrated GPU power and performance model. In: Proceedings of the 37th annual international symposium on Computer architecture (ISCA '10). ACM, New York, NY, USA: 280-289.
67. [Houzeaux2017] Houzeaux G., Codina R. (2017). A Geometrical Domain Decomposition Method in Computational Fluid Dynamics, Monographs of the International Centre for Numerical Methods in Engineering (CIMNE), https://www.scipedia.com/public/Houzeaux_Codina_2017a
68. [Houzeaux2018] Houzeaux G., Borrell R., Fournier Y., Garcia- Gasulla M., Göbbert J.H., Hachem E., Mehta V., Mesri Y., Owen H., Vázquez M. (2018). High-Performance Computing: Dos and Don'ts. In: Computational Fluid Dynamics, Basic Instruments and Applications in Science, A. Ionescu (Ed.): 3-42. Intech, 2018 http://dx.doi.org/10.5772/intechopen.72042.
69. [Hugo2013] Hugo A. E., Guermouche A., Wacrenier P A., Namyst R. (2013). Composing Multiple StarPU Applications over Heterogeneous Machines: A Supervised Approach. 2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum, Cambridge, MA: 1050-1059.
70. [Hou1994] Hou E., Ansari N., Ren H. (1994). A genetic algorithm for multiprocessor scheduling, IEEE Trans. Parallel Distrib. Syst., 5 (2): 113-120.
71. [Hu2016] Hu L., Che X., Zheng S.-Qi. (2016). A Closer Look at GPGPU. ACM Computing Surveys. 48, 4, Article 60 (March 2016), 20 pages. DOI: https://doi.org/10.1145/2873053.
72. [Ibrahim2012] Ibrahim D. (2012). Teaching microcontroller programming using "TINA" simulation. AWER Procedia Information Technology Computer Science 1: 42-47.
73. [Iverson1999] Iverson M., Özgüner F., Potter L. C. (1999). Statistical Prediction of Task Execution Times Through Analytic Benchmarking for Scheduling in a Heterogeneous Environment. Proceedings of the Eighth Heterogeneous Computing Workshop: 1374 - 1379.
74. [Jackson2012] Jackson M., Budruk R. (2012). PCI Express Technology Comprehensive Guide to Generations 1.x, 2.x, 3.0 , MindShare.
75. [Jahr2012] Jahr R., Calborean H., Vintan L., Ungerer T. (2012). Finding Near-Perfect Parameters for Hardware and Code Optimizations with Automatic Multi-Objective Design Space Explorations. Concurrency and Computation: Practice and Experience, John Wiley & Sons, 27 (9): 2196-2214. doi: 10.1002/cpe.2975
76. [Jeannot2006] Jeannot E., Vernier F. (2006). A Practical Approach of Diffusion Load Balancing Algorithms. [Research Report] RR-5875, INRIA: 20 p. https://hal.inria.fr/inria-00071394/document.
77. [Jensen2009] Jensen K., Kristensen L.M. (2009|). Coloured Petri Nets, Springer-Verlag Berlin Heidelberg.
78. [Jouault2005] Jouault F., Kurtev I. (2005). Transforming Models with AT. In: MoDELS 2005 Workshops, J.-M. Bruel (Ed.), LNCS 3844: 128.
79. [Kale2013] Kale L.V., Bhatele A. |(2013). Parallel Science and Engineering Applications: The Charm++ Approach (1st ed.). CRC Press, Inc., Boca Raton, FL, USA.

80. [Kanemitsu2016] Kanemitsu H., Hanada M., Nakazato H. (2016). Clustering-Based Task Scheduling in a Large Number of Heterogeneous Processors. In: IEEE Transactions on Parallel and Distributed Systems, 27, (11): 3144-3157.
81. [Kanungo2016] Kanungo P. (2016). Scheduling in Distributed Computing Environment Using Dynamic Load Balancing, Anchor. Academic Publishing.
82. [Kasmi2017] Kasmi N. Zbakh M., Samadi Y., Cherkaoui R., Haouari A. (2017). Performance evaluation of StarPU schedulers with preconditioned conjugate gradient solver on heterogeneous (multi-CPUs/multi-GPUs) architecture. 2017 3rd International Conference of Cloud Computing Technologies and Applications (CloudTech), Rabat: 1-6.
83. [Kaur2017] Kaur N., Chhabra A. (2017). Comparative Analysis of Job Scheduling Algorithms in Parallel and Distributed Computing Environments. International Journal of Advanced Research in Computer Science, 8 (3): 948-956.
84. [Kaw2011] Kaw A. K., Kalu E. K., Nguyen D. (2011). Numerical Methods with Applications. Textbooks Collection11.http://scholarcommons.usf.edu/oa_textbooks/11.
85. [Khan2017] Khan S., Nazir B., Khan I. A., Shamshirband S., Chronopoulos A. T. (2017). Load balancing in grid computing: Taxonomy, trends and opportunities. Journal of Network and Computer Applications, 88: 99-111.
86. [Khokhar1993] Khokhar A.A., Prasanna V. K., Shaaban M. E., Wang C.-L. (1993). Heterogeneous Computing: Challenges and Opportunities. IEEE Computer, IEEE Computer Society Press Los Alamitos, CA, USA, 26 (6):18-27.
87. [Khouadjia2013] Khouadjia M.R., Schoenauer M., Vidal V., Dréo J., Savéant P. (2013) Multi-objective AI Planning: Comparing Aggregation and Pareto Approaches. In: Middendorf M., Blum C. (eds) Evolutionary Computation in Combinatorial Optimization. EvoCOP 2013. Lecture Notes in Computer Science, vol 7832. Springer, Berlin, Heidelberg
88. [Kienhuis1998] Kienhuis B., Deprettere E., Vissers K., van der Wolf P. (1998). The construction of a retargetable simulator for an architecture template. Proceedings of 6-th Int. Workshop on Hardware/Software Codesign, Seattle,Washington: 5 p. https://ptolemy.berkeley.edu/~kienhuis/ftp/codes98.pdf
89. [Kim2007] Kim J., Dally W., Abts D. (2007). Flattened butterfly: a cost-efficient topology for high-radix networks.. ACM Sigarch Computer Architecture News. 35. 126-137.
90. [Kim2008] Kim J., Dally W. J., Scott S., Abts D. (2008). Technology -Driven, Highly - Scalable Dragonfly Topology. Proceedings of the 35th International Symposium on Computer Architecture: 1-3.
91. [Kjolstad2010] Kjolstad F., Snir M. (2010). Ghost Cell Pattern. 2nd Annual Workshop on Parallel Programming Patterns: 9 p. http://people.csail.mit.edu/fred/ghost_cell.pdf
92. [Kohl2016]Kohl J., Reichenbach M., Demel C., Fey D., A SystemC Based Framework for Cycle Accurate Processor Simulation and Parameter Analysis, IFAC-PapersOnLine,Volume 49, Issue 25, 2016,Pages 92-97,https://doi.org/10.1016/j.ifacol.2016.12.016.
93. [Kounev2010] Kounev S., Spinner S., Meier P. (2010). QPME 2.0 - A Tool for Stochastic Modeling and Analysis using Queueing Petri Nets. In: Active Data Management: From active databases to event-based systems and more Pablo Guerrero, I. Petrov, and K.Sachs (eds.), Springer.
94. [Krüger2016] Krüger T., Kusumaatmaja H., Kuzmin A., Shardt O., Silva G., Viggen E. M. (2016). The Lattice Boltzmann Method - Principles and Practice, Springer.
95. [Kunzman2011] Kunzman D.M., Kale L.V. (2011). Programming Heterogeneous Clusters with Accelerators using Object-Based Programming. Journal of Scientific Programming, IOS Press, 19 (1): 47–62.
96. [Kunzman 2009] Kunzman D., Kale, L. (2009). Towards a framework for abstracting accelerators in parallel applications: experience with CellBE. SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, Portland, Oregon: 1-12. http://doi.acm.org/10.1145/1654059.1654114
97. [Lagravière2016] Lagravière J., J. Langguth M. Sourouri P., Ha H., Cai X. (2016). On the performance and energy efficiency of the PGAS programming model on multicore architectures. 2016 International Conference on High Performance Computing & Simulation (HPCS), Innsbruck: 800-807.
98. [Leiserson1985] Leiserson C.E. (1985). Fat-trees: universal networks for hardware-efficient supercomputing, IEEE Transactions on Computers, 34 (10): 892-901.

99. [Levesque 2013] Levesque M., Larkin, J. (2013). Refactoring Applications for the XK7 and Future Hybrid Architectures. https://www.slideshare.net/jefflarkin/refactoring-applications-for-the-xk7-and-future-hybrid-architectures
100. [Li2009] Li S., Ahn J. H., Strong R. D., Brockman J. B., Tullsen D. M., Jouppi N. P. (2009). McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), New York, NY: 469-480.
101. [Li2015] Li P. (2015). Unified system of code transformation and execution for heterogeneous multi-core architectures. Bordeaux, PhD Thesis. https://tel.archives-ouvertes.fr/tel-01342119/document
102. [Li2016] Li S., Huang P-C, Banks D., DePalma M.., Elshaarany A., Hemmert K. S.., Rodrigues A., Ruppel E., Wang Y., Ang J., Jacob B. (2016). Low Latency, High Bisection-Bandwidth Networks for Exascale Memory Systems. Proceedings of the Second International Symposium on Memory Systems MEMSYS 2016: 62-73.
103. [Liao2013]Liao C., Yan Y., de Supinski B.R., Quinlan D.J., Chapman B. (2013). Early Experiences with the OpenMP Accelerator Model. In: OpenMP in the Era of Low Power Devices and Accelerators, Rendell A.P., Chapman B.M., Müller M.S. (eds.). IWOMP 2013. Lecture Notes in Computer Science, vol 8122. Springer, Berlin, Heidelberg
104. [Marranghello2012] Marranghello N., De Oliveira W. L. A., Damiani F. (2012). On The Use Of Petri Nets For The Description Of Digital Systems. Proceedings of Brazilian Petri Net Meeting 2012, Natal, Brazil http://www.dcce.ibilce.unesp.br/~norian/publicacoes/bpnm2002.pdf
105. [Martin2012] Martin M.O., Computer Simulation in Physics and Engineering, de Gruyter,2012
106. [Martinez2009] Martinez D. R., Cabaleiro J. C., Pena T. F., Rivera F. F., Blanco V. (2009). Accurate analytical performance model of communications in MPI applications. 2009 IEEE International Symposium on Parallel & Distributed Processing, Rome: 1-8.
107. [Mehta2014] Mehta M., Jinwala D. (2014). A Hybrid Dynamic Load Balancing Algorithm for Distributed System. International Journal of Distributed Systems and Technologies, 9(8): 1825-1833.
108. [Mei2012] Mei J., Li K. (2012). Energy-Aware Scheduling Algorithm with Duplication on Heterogeneous Computing Systems. Proceedings of the 2012 ACM/IEEE 13th International Conference on Grid Computing, September 20-23: 122-129.
109. [Meng2009] Meng J., Skadron K. (2009). Performance Modeling and Automatic Ghost Zone Optimization for Iterative Stencil Loops on GPUs, ICS '09: Proceedings of the 23rd International Conference on Supercomputing: 256-265.
110. [Min2011] Min S.-J., Iancu C., Katherine Y. (2011). Hierarchical work stealing on manycore clusters. Fifth Conference on Partitioned Global Address Space Programming Models (PGAS): 10 p. http://upc.lbl.gov/publications/MinEtAl-HotSLAW-Work-Stealing-PGAS11.pdf
111. [Mironescu2011] **Mironescu I. D.**, Vintan L. (2011). Optimally Mapping a CFD Application on a HPC Architecture. ACACES 2011 Seventh International Summer School on Advanced Computer Architecture and Compilation for High-Performance and Embedded Systems, ISBN 978 90 382 1798 7, Published by FP7 HiPEAC Network of Excellence, 10-16 July 2011, Fiuggi, Italy: 227-230.
112. [Mironescu2013] **Mironescu I. D.**, Vintan L. (2013). Performance prediction for parallel applications running on HPC architectures through Petri net modelling and simulation. Intelligent Computer Communication and Processing (ICCP), 2013 IEEE Conference Publications: 267 – 270. DOI: 10.1109/ICCP.2013.6646119.
113. [Mironescu2014] **Mironescu I. D.**, Vinţan L. (2014). Coloured Petri Net modelling of task scheduling on a heterogeneous computational node. IEEE International Conference Intelligent Computer Communication and Processing (ICCP): 323-330.
114. [Mironescu2017] **Mironescu I. D**., Vinţan L. (2017). A task scheduling algorithm for HPC applications using colored stochastic Petri Net models. ICCP 2017: 479-486.
115. [Mironescu2018] **Mironescu I. D**., Vinţan L. (2018). A Simulation Based Analysis of an Multi Objective Diffusive Load Balancing Algorithm, International Journal of Computers, Communications & Control, 13 (4): 503 – 520.
116. [Muralimanohar2009] Muralimanohar N., Balasubramonian R., Jouppi N. P. (2009). CACTI 6.0: A Tool to Understand Large Caches. Technical Report HPL-2009-85, http://www.hpl.hp.com/techreports/2009/HPL-2009-85.html

117. [Nayak 2012] Nayak S. K., Padhy S. K., Panigrahi S. P. (2012). A novel algorithm for dynamic task scheduling. Future Generation Computer Systems, 28 (5): 709-717.
118. [Palmer2016] Palmer TN. (2016). A personal perspective on modelling the climate system. Proceedings of the Royal Society A 472: 20150772. http://dx.doi.org/10.1098/rspa.2015.0772
119. [Pearce2016] Pearce O., Gamblin T., de Supinski B., Amato N. (2016). MPMD Framework for Offloading Load Balance Computation. In: Proc. Int. Par. and Dist. Proc. Symp. (IPDPS), Chicago, IL, USA; 943-952. file:///C:/Users/WIN10/Downloads/olga-ipdps2016.pdf
120. [Perez2010] Perez-Palacin D., Merseguer J. (2010). Performance Evaluation of Self-reconfigurable Service-oriented Software with Stochastic Petri Nets. Electr. Notes Theor. Comput. Sci. 261: 181-201.
121. [Phillips2014] Phillips J.C., Stone J.E., Vandivort K.L., Armstrong T.G., Wozniak M.J., Wilde M., Schulten K. (2014). Petascale Tcl with NAMD, VMD, and Swift/T. 2014 First Workshop for High Performance Technical Computing in Dynamic Languages, pp. 6-17, doi: 10.1109/HPTCDL.2014.7.
122. [Piotrowski2017] Piotrowski A. P., Napiorkowski M.J., Napiorkowski J.J., Rowinski P.M. (2017). Swarm Intelligence and Evolutionary Algorithms: Performance versus speed. Information Sciences, 384: 34-85. https://doi.org/10.1016/j.ins.2016.12.028.
123. [Popova2013] Popova-Zeugmann, L. (2013). Time and Petri Nets, Springer.
124. [Pozrikidis2016] Pozrikidis C. (2016). Fluid Dynamics: Theory, Computation and Numerical Simulation, Third Edition, Springer.
125. [Raicu2014] Raicu I., Palur S. (2014). Understanding Torus Network Performance through Simulations, Greater Chicago Area System Research Workshop (GCASR).
126. [Rauber2014] Rauber T., Rünger G., Schwind M., Xu H., Melzner S. (2014). Energy measurement, modeling, and prediction for processors with frequency scaling. The Journal of Supercomputing, 70: 1451-1476.
127. [Regier2018] Regier J., Pamnany K., Fischer K., Noack A., Lam M., Revels J., Howard S., Giordano R., Schlegel D., McAuliffe J., Thomas R., Kumar P. (2018). Cataloging the Visible Universe through Bayesian Inference at Petascale. Accepted to IPDPS 2018, E-print arXiv:1801.10277.
128. [Reisig2013] Reisig W. (2013). Understanding Petri Nets: Modeling Techniques, Analysis Methods, Case Studies, Springer.
129. [Ribeiro2015] Ribeiro R., Barbosa J., Santos L.P. (2015). A framework for efficient execution of data parallel irregular applications on heterogeneous systems. Parallel Processing Letters, 25 (2): 30 p.
130. [Robson2016] Robson M.P., Buch R., Kale L.V. (2016). Runtime Coordinated Heterogeneous Tasks in Charm++. Proceedings of the Second International Workshop on Extreme Scale Programming Models and Middleware, ESPM2, IEEE Press: 40-43, https://doi.org/10.1109/ESPM2.2016.7 .
131. [Rogers2016] Rogers P. (2016). Chapter 2 - HSA Overview. Heterogeneous System Architecture, Wen-mei W. Hwu, (Ed.), Morgan Kaufmann, ISBN 9780128003862: 7-18. https://doi.org/10.1016/B978-0-12-800386-2.00001-8.
132. [Rust2003] Rust C., Rettberg A., Gossens K., "From high-level Petri nets to SystemC," SMC'03 Conference Proceedings. 2003 IEEE International Conference on Systems, Man and Cybernetics. Conference Theme - System Security and Assurance (Cat. No.03CH37483), Washington, DC, 2003, pp. 1032-1038 vol.2.
133. [Said2017] Said I., Fortin P., Lamotte J.L., Calandra H. (2017). Leveraging the accelerated processing units for seismic imaging: A performance and power efficiency comparison against CPUs and GPUs. The International Journal of High Performance Computing Applications. https://doi.org/10.1177/1094342017696562.
134. [Saini2017] Saini S., Hood R. (2017). Performance Evaluation of Intel Broadwell Nodes Based Supercomputer Using Computational Fluid Dynamics and Climate Applications. 2017 IEEE 19th International Conference on High Performance Computing and Communications Workshops (HPCCWS), Bangkok, pp. 58-65. doi: 10.1109/HPCCWS.2017.00015.
135. [Savage2008] Savage J.E., Zubair M. (2008). A unified model for multicore architectures. In: Proc. 1st International Forum on Next-Generation Multicore/Manycore Technologies. 13 p. http://cs.brown.edu/people/jsavage/papers/IFMT08.pdf
136. [Schmidt2018] Schmidt B., González-Domínguez J., Hundt Ch., Schlarb M. (2018). Chapter 10 - Unified Parallel C++. In: Parallel Programming, Schmidt B., González-Domínguez J., Hundt Ch., Schlarb M. (eds.), Morgan Kaufmann: 365-398.

137.[Shterenlikht2013] Shterenlikht A. (2013). Fortran co-array library for 3D cellular automata microstructure simulation. Proc. 7th PGAS Conf., Edinburgh.
138.[Shterenlikht2015] Shterenlikht A., Margetts L., Cebamanos L., Henty D. (2015). Fortran 2008 coarrays. ACM SIGPLAN Fortran Forum, 34:10-30. DOI: 10.1145/2754942.2754944.
139.[Shterenlikht2018] Shterenlikht A., Margetts L., Cebamanos L. (2018). Modelling fracture in heterogeneous materials on HPC systems using a hybrid MPI/Fortran coarray multi-scale CAFE framework. Adv. Eng. Software. DOI: 10.1016/j.advengsoft.2018.05.008.
140.[Sirisha2016] Sirisha D., Vijayakumari G. (2016). Exploring the Efficacy of Branch and Bound Strategy for Scheduling Workflows on Heterogeneous Computing Systems. Procedia Computer Science, 93:315-323. https://doi.org/10.1016/j.procs.2016.07.216.
141.[Smith1990] Smith C. U. (1990). Performance engineering of software systems. The SEI Series in Software Engineering, Addison–Wesley.
142.[Steinberg2008] Steinberg D., Budinsky F., Paternostro M., Merks E. (2008). EMF: Eclipse Modeling Framework. Addison-Wesley Professional.
143.[Storti2015] Storti D., Yurtoglu M. (2015). CUDA for Engineers: An Introduction to High-Performance Parallel Computing (1st ed.). Addison-Wesley Professional.
144.[Tang2010] Tang X., Li K., Liao G., Li R. (2010). List scheduling with duplication for heterogeneous computing systems. Journal of Parallel and Distributed Computing, 70(4): 323–329.
145.[Tchiboukdjian2011] Tchiboukdjian M., Gast N., Trystram D. (2011). Decentralized List Scheduling. Annals of Operations Research. 207: 1-21. DOI 10.1007/s10479-012-1149-7.
146.[Tian2014] Tian M., Gu W., Pan J., Guo M. (2014). Performance Analysis and Optimization of PalaBos on Petascale Sunway BlueLight MPP Supercomputer. In: Parallel Computational Fluid Dynamics. Series Communications in Computer and Information Science, Li K., Xiao Z., Wang Y., Du J., and Li K. (eds.). volume 405, Springer Berlin Heidelberg: 311-320.
147.[Thoman2011] Thoman P., Kofler K., Studt H., Thomson J., Fahringer T. (2011). Automatic OpenCL Device Characterization: Guiding Optimized Kernel Design. Euro-Par 2011 Parallel Processing, Lecture Notes in Computer Science 6853, , Springer Berlin Heidelberg: 438-452.
148.[Tomasz2012] Tomasz R., Werewka J. (2012). Performance Analysis of Interactive Internet Systems for a Class of Systems with Dynamically Changing Offers. Advances in Software Engineering Techniques, Lecture Notes in Computer Science, 7054: 109-123.
149.[Topcoglu2002] Topcuoglu H., Hariri S., Wu M.-Y. (2002). Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Transactions on Parallel and Distributed Systems, 13(3): 260–274.
150.[Ubal2012] Ubal R., Jang B., Mistry P., Schaa D., Kaeli D. (2012). Multi2Sim: a simulation framework for CPU-GPU computing. Proceedings of the 21st international conference on Parallel architectures and compilation techniques (PACT '12). ACM, New York, NY, USA: 335-344.
151.[Valiev2010] Valiev M., Bylaska E.J., Govind N., Kowalski K., Straatsma T.P., van Dam H.J.J., Wang D., Nieplocha J., Apra E., Windus T.L., de Jong W.A. (2010). NWChem: a comprehensive and scalable open-source solution for large scale molecular simulations. Comput. Phys. Commun. 181: 1477-1489.
152.[vanWerkhoven2014] van Werkhoven B.V., Maassen J., Seinstra F.J., Bal H.E. (2014). Performance Models for CPU-GPU Data Transfers. 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing: 11-20.
153. [Vin00] Vinţan N. Lucian – Arhitecturi de procesoare cu paralelism la nivelul instrucțiunilor, Editura Academiei Române, București, ISBN 973-27-0734-8, 2000
154. [Vin07] Vinţan N. Lucian – Prediction Techniques in Advanced Computing Architectures, Matrix Rom Publishing House, Bucharest, ISBN 978-973-755-137-5, 2007
155. [Vin16] Vinţan N. Lucian – Fundamente ale arhitecturii microprocesoarelor, Editura Matrix Rom, București, ISBN 978-606-25-0276-8, 2016
156. [Vin16b] Vinţan N. Lucian et al. – Improving Computing Systems Automatic Multi-Objective Optimization through Meta-Optimization, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, ISSN: 0278-0070, Volume 35, Issue 7, pp. 1125-1129, July 2016

157. [Vizitiu2014] Vizitiu A., Itu L.M., Nita C., Suciu C. (2014). Optimized three-dimensional stencil computation on Fermi and Kepler GPUs. 2014 IEEE High Performance Extreme Computing Conference (HPEC): 1-6.
158. [Vu2015] Vu L.-H., Foures D., Albert V. (2015). ProDEVS: an Event-Driven Modeling and Simulation Tool for Hybrid Systems using State Machines. Proceedings of the 8th International Conference on Simulation Tools and Techniques (8): 29-37.
159. [Wadekar2013] Wadekar M. (2013). Chapter 11 - InfiniBand, iWARP, and RoCE., Handbook of Fiber Optic Data Communication. In Casimer DeCusatis (ed). Academic Press: 267-287. https://doi.org/10.1016/B978-0-12-401673-6.00011-8.
160. [Wang2014] Wang C., Feng X., Li X., Zhou X., Chen P. (2014), Colored Petri Net Model with Automatic Parallelization on Real-Time Multicore Architectures. Journal of Systems Architecture, 60 (3): 293-304.
161. [Wang2016] Wang G., Wang Y., Liu H., Guo H. (2016). HSIP: A Novel Task Scheduling Algorithm for Heterogeneous Computing. Scientific Programming, Article ID 3676149, 11 pages. https://doi.org/10.1155/2016/3676149.
162. [Wei 2015] Wei J., Xu D., Qin Y. (2015). A heuristic algorithm for solving the problem of load balancing. Proceedings of the Seventh International Conference on Advanced Computational Intelligence (ICACI), Wuyi, China.
163. [Wells2006] Wells L. (2006). Performance analysis using CPN tools. Proceedings of the 1st international conference on Performance evaluation methodolgies and tools (valuetools '06). ACM, New York: 1-10. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.105.3766&rep=rep1&type=pdf
164. [Werner2011] Werner H.-J., Knowles P. J., Knizia G., Manby F. R., Schütz M. (2011). Molpro - a general purpose quantum chemistry program package. WIRE Computational Molecular Science, 2 (2): 242-253
165. [Winsberg2010] Winsberg E. (2010). Science in the Age of Computer Simulation, University Of Chicago Press, Chicago.
166. [Woodgate2018] Woodgate M. A., Barakos G. N., Steijl R., Pringle G. J. (2018). Parallel performance for a real time Lattice Boltzmann code. Computers & Fluids, 173: 237-258, https://doi.org/10.1016/j.compfluid.2018.03.004.
167. [XuY2014] Xu Y., Li K., Hu J., Li K. (2014). A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues. Information Sciences, 270: 255-287. https://doi.org/10.1016/j.ins.2014.02.122.
168. [XuS2014] Xu S., Guo Z., Hu G., Chen W., Lewis R., Wong C.-N. (2014). Thermal and Flow Fields. In Single Board Computer Cabin Systems Using CFD Analysis. Engineering Applications of Computational Fluid Mechanics, 8(4): 574-585. DOI:10.1080/19942060.2014.11083308.
169. [Yazdanpanah2015] Yazdanpanah F., Álvarez C., Jiménez-González D., Badia R., Valero M. (2015). Picos: A hardware runtime architecture support for OmpSs. Future Generation Computer Systems, 53: 130-139.
170. [Yu2018] Yu S., Li K., Xu Y. (2018). A DAG task scheduling scheme on heterogeneous cluster systems using discrete IWO algorithm. Journal of Computational Science, 26: 307-317. https://doi.org/10.1016/j.jocs.2016.09.008.
171. [Zheng 2014] Zheng Y., Kamil A., Driscoll M., Shan H., Yelick K. (2014). UPC++: A PGAS Extension for C++. 28th IEEE International Parallel and Distributed Processing Symposium: 1-10. https://people.eecs.berkeley.edu/~driscoll/pdfs/ipdps2014.pdf
172. [Zheng2010] Zheng G., Gupta G., Bohm E., Dooley I., Kale L.V. (2010). Simulating Large Scale Parallel Applications using Statistical Models for Sequential Execution Blocks. Proceedings of the 16th International Conference on Parallel and Distributed Systems (ICPADS 2010) Shanghai, China: 10-15.
173. [Zimmerman2014] Zimmerman J., Spurgeon C. E. (2014). Ethernet: The Definitive Guide,O'Reilly Media.
174. [acceleo] *** https://www.eclipse.org/acceleo/
175. [amdocl]*** http://developer.amd.com/wordpress/media/2013/12/AMD_OpenCL_Programming_Optimization_Guide2.pdf
176. [codexl]***CodeXL – Powerful Debugging, Profiling & Analysis http://developer.amd.com/tools-and-sdks/opencl-zone/opencl-tools-sdks/codexl/
177. [clay]*** http://www.claymath.org/millennium-problems
178. [coka]*** http://www.fe.infn.it/coka/doku.php?id=start retrived february 2018

179. [eclipse] *** https://www.eclipse.org/
180. [fluent] *** https://www.ansys.com/products/fluids/ansys-fluent
181. [fermi] *** https://en.wikipedia.org/wiki/Fermi_(microarchitecture)
182. [hpcwshp]***]http://www.hpcadvisorycouncil.com/events/2012/Switzerland-Workshop/Presentations/Day_3/7_Simula.pdf
183. [intel2012] Intel® 64 and IA-32 Architectures Optimization Reference Manual, http://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-optimization-manual.html (04.2012)
184. [ib_melanox]*** Introduction to InfiniBand, white paper https://www.mellanox.com/pdf/whitepapers/IB_Intro_WP_190.pdf
185. [pop]*** POP Standard Metrics for Parallel Performance Analysis - POP CoE https://pop-coe.eu/node/69 Performance Optimisation and Productivity (POP) A Centre of Excellence (CoE) in Computing Applications BSC-CNS | Barcelona Supercomputing Center retrieved march 2018
186. [powerflow] https://exa.com/en/product/simulation-tools/powerflow-cfd-simulation
187. [omnipath] *** Intel® Omni Path Architecture.
188. http://calcul.math.cnrs.fr/IMG/pdf/intel-omni-path-for-journees-mesocentre.pdf
189. [openfoam] *** https://www.openfoam.com/
190. [pragma] *** http://www.pragmadev.com/
191. [sis2013] *** Benchmarks : Measuring Cache and Memory Latency (access patterns, paging and TLBs) http://www.sisoftware.net/?d=qa&f=ben_mem_latency (24.04.2013)
192. [sniper]*** http://snipersim.org/w/The_Sniper_Multi-Core_Simulator
193. [star] *** https://mdx.plm.automation.siemens.com/star-ccm-plus
194. [stratix]***https://www.top500.org/news/german-university-will-deploy-fpga-powered-cray-supercomputer/
195. [tina] *** https://www.tina.com/
196. [top500] *** https://www.top500.org
197. [upc] *** **U**PC Language and Library Specifications, v1.3. UPC Consortium, Lawrence Berkeley National Lab Tech Report LBNL-6623E, Nov 2013, https://escholarship.org/uc/item/0n7734mg
198. [uperf] *** http://www.uperf.org/
199. [xeonphi] *** https://software.intel.com/en-us/articles/using-intel-mpi-library-on-intel-xeon-phi-product-family

ANEXA. Lista lucrărilor publicate de autor pe tematica tezei de doctorat

1. **MIRONESCU I. D.**, VINŢAN L. (2011). *Optimally Mapping a CFD Application on a HPC Architecture*. ACACES 2011 Seventh International Summer School on Advanced Computer Architecture and Compilation for High-Performance and Embedded Systems, Poster Abstracts, pp. 227-230, ISBN 978 90 382 1798 7, Published by FP7 HiPEAC Network of Excellence, 10-16 July 2011, Fiuggi, Italy

2. **MIRONESCU I. D.**, VINŢAN L. (2013). *Performance Prediction for Parallel Applications Running on HPC Architectures through Petri Net Modelling and Simulation*. 9th International Conference on Intelligent Computer Communication and Processing (ICCP 2013), ISBN 978-1-4799-1493-7, pp. 276-270, IEEE Computer Society Press, Cluj-Napoca, September 5 - 7 2013 (*IEEE Xplore*), http://www.iccp.ro/iccp2013/. Indexat IEEE, v. http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=6646119&url=http%3A%2F%2Fieeexplore.ieee.org%2Fiel7%2F6636285%2F6646058%2F06646119.pdf%3Farnumber%3D6646119; 57 accepted papers from a total of 91 submitted papers. DOI: 10.1109/ICCP.2013.6646119

3. **MIRONESCU I. D.**, VINŢAN L. (2014). *Colored Petri net modelling of task scheduling on a heterogeneous computational node*, Proceedings of 10th International Conference on Intelligent Computer Communication and Processing (ICCP 2014), ISBN 978-1-4799-6568-7, pp. 323-330, IEEE Computer Society Press, Cluj-Napoca, September 4 - 6 2014 (IEEE Xplore - http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=6937016&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D6937016), v. http://www.iccp.ro/iccp2014/; *full paper;* **indexată ISI Thomson Reuters Proceedings Web of Science.** WOS:000348677300048

4. **MIRONESCU I. D.**, VINŢAN L. (2017). *A task scheduling algorithm for HPC applications using colored stochatic Petri net models*. Proceedings of 13th International Conference on Intelligent Computer Communication and Processing (ICCP 2017), ISBN 978-1-5386-3368-7/17, IEEE Computer Society Press, Cluj-Napoca, September 7 - 9 2017, v. https://www.researchgate.net/publication/321149974_A_Task_Scheduling_Algorithm_for_HPC_Applications_using_Colored_Stochastic_Petri_Net_Models; **Articol indexat (ISI) Thomson Reuters Web of Science (WoS) Proceedings**, v. http://apps.webofknowledge.com/Search.do?product=WOS&SID=Z26mRYgKg5tnR4BIMKy&search_mode=GeneralSearch&prID=9593451c-dad9-4766-be18-36d3d05524eb

5. **MIRONESCU I. D.,** VINŢAN L. (2018). *A simulation based analysis of an multi objective diffusive load balancing algorithm*. International Journal of Computers, Communications & Control, pp. 503 - 520, ISSN 1841–9836, Volume 13, Issue 4, 2018, **Revista este cotată Clarivate Analytics Web of Science, Impact Factor: IF=1,290**, AIS=0,170, cf. JCR 2017, publicat în iunie 2018, Q3 (98/148) in Computer Science, v. http://jcr.incites.thomsonreuters.com.am.e-nformation.ro/JCRJournalProfileAction.action?Init=Yes&tab=RANK&issn=1841-9836&SrcApp=IC2LS&SID=H4-mcp4dRPpRpIQPceGpmUCcSDM1lx2F8fjnG-18x2dQM6hosx2Bnqyeix2BC4bxxs5fRAx3Dx3DgdvnEp3lhW7Zp5MdtfpNCAx3Dx3D-9vvmzcndpRgQCGPd1c2qPQx3Dx3D-wx2BJQh9GKVmtdJw3700KssQx3Dx3D

Două dintre aceste lucrări au fost citate în publicațiile:

- Cheng, Y., Zhao, S., Cheng, B., Hou, S., Shi, Y., & Chen, J., *Modeling and Optimization for Collaborative Business Process Towards IoT Applications*, Mobile Information Systems, ISSN: 1574-017X, Accepted 8 August 2018 (v. https://www.hindawi.com/journals/misy/aip/9174568/, https://www.hindawi.com/journals/misy/aip/), v. https://scholar.google.com/scholar?oi=bibs&hl=ro&cites=17875923396319682995 (revistă cotată WoS, v. http://apps.webofknowledge.com.am.e-nformation.ro/Search.do?product=WOS&SID=D4If9B4JXDdZQp7QdkF&search_mode=GeneralSearch&prID=cf6bea33-3de2-43eb-bdce-d91675fe2d21)

- Li Z. et al, *Performance analysis for job scheduling in hierarchical HPC systems: A coloured petri nets method*, 15-th International Conference on Algorithms and Architectures for Parallel Processing, ICA3PP 2015; Zhangjiajie; China; 18 November 2015 -20 November 2015, Volume 9531, 2015, Pages 259-280, v. https://www.scopus.com/record/display.uri?eid=2-s2.0-84951764168&origin=resultslist&sort=plf-f&cite=2-s2.0-84891126159&src=s&imp=t&sid=173CDD260702332F36FF071584B23048.wsnAw8kcdt7IPYLO0V48gA%3a60&sot=cite&sdt=a&sl=0&relpos=0&citeCnt=0&searchTerm=